Secure and High Performance Volunteer Computing Platform

TOHOKU UNIVERSITY

Graduate School of Information Sciences

Secure and High Performance Volunteer Computing Platform
(安全で高性能なボランティアコンピューティング基盤に関する研究)

A dissertation submitted for the degree Doctor of Philosophy (Information Sciences)

Department of Computer and Mathematical Sciences

by

Hong WANG

16 January 2009

Secure and High Performance Volunteer Computing Platform

Hong Wang

Abstract

This dissertation discusses a secure and high performance volunteer computing platform. This computing platform supports more computational problems, guarantees lower performance degradation caused by task failure, compared to the existing volunteer computing platforms. It also enables secure volunteer data processing, using a hardware-based cryptography method.

Volunteer computing platforms are used to solve large-scale computational problems with the computing power from the Internet-connected individual computers, game consoles, and other computing resources. There have been three successful pioneering volunteer computing projects: GIMPS (The Great Internet Mersenne Prime Search), Distributed.net, and SETI@home. Inspired by these projects, several well-known volunteer computing platforms such as Folding@home, BOINC (Berkeley Open Infrastructure for Network Computing), XtremWeb, Entropia, Alchemi, and JNGI are designed and applied to solve various computational problems. By December 2008, the most powerful volunteer computing platform - Folding@ achieved more than four Petaflops computing power by connecting more than 3,500,000 CPUs. In contrast, the fastest supercomputer, Roadrunner achieves a maximum LINPACK performance of 1.026 Petaflops.

Despite the massive computing power offered by the existing volunteer computing platforms, there are two major issues that limit the applicable areas. One issue is the lack of support for various computational models. So far, volunteer computing platforms can only solve embarrassingly parallel problems that can obviously be divided into a number of completely independent tasks. Because it is only minority of all the computational problems, inter-task dependency support is required to solve complex problems on volunteer computing platforms. The inter-task dependency may result in serious performance degradation, as a result of the frequent peer failure and the waste of computing power while no task can be dispatched because of dependency. Thus, it also requires optimized task dispatch policies to guarantee low performance degradation. The other issue is the lack of support for sensitive data processing. Therefore, the volunteer computing has only been applied to process publicly available data, such as the radio telescope data in the SETI@home project. To process the sensitive data such as medical data, biology data, market research data, and financial data on volunteer computing platforms, security features are strongly desired.

These issues exist on different layers of the volunteer computing platforms, including task management layer, resource management layer, and the application layer. To

solve them, several solutions on different layer are required to work together. To support inter-task dependency, a workflow management mechanism is introduced on the task management layer. It controls and maintains the workflow at the runtime. To mitigate the performance degradation introduced by the inter-task dependency, optimized task dispatch policies are proposed on the resource management layer. The optimized task dispatch requires runtime information such as worker's availability data. Therefore, worker availability checking is also added to this layer. On the application layer, a cryptography-based secure data processing method that involves both the dispatcher and the worker is proposed to support secure data processing. The dissertation studies these solutions in the following three chapters.

First, a dependable workflow management mechanism is presented. Before presenting this mechanism, the related work is reviewed. So far, no workflow management mechanism exists for volunteer computing platforms. This is because of the fact that volunteer computing platforms are mostly used for solving embarrassingly parallel problems and not for other kinds of computational problems. Workflow management mechanisms have been studied a lot for grid systems. None of them can be applied to the volunteer computing platform because these grid workflow systems focus on finding optimized scheduling/mapping of tasks. In the case of the volunteer computing, the performance degradation for frequent failures needs to be considered first. The dependable workflow management mechanism includes a workflow management mechanism, a redundant task dispatch and a runtime optimization for the redundant task dispatch. The workflow management mechanism consists of a workflow markup language and a workflow management module for the dispatcher. The dispatcher also needs to consider the overheads caused by task failures, because the tasks must wait for the preceding tasks' results if a volunteer computing system with volatile peers supports inter-task dependency. Therefore, a redundant task dispatch mechanism is proposed to reduce the overheads. Redundant task dispatch uses the computing power from the idle workers to process duplicate copies of tasks, thus the task failure probability can be reduced. Therefore, the performance degradation can be reduced. However, unlimited redundant task dispatch may result in performance degradation, because too much computing power is wasted for the processing of duplicate copies. As the optimal value for the number of redundant tasks depends on the runtime parameters that are unknown in a real computing environment, a runtime optimization is designed for the redundant task dispatch to find the optimal value. The performance of the proposed mechanisms is evaluated using a simulator. The large-scale simulation results indicate that the redundant task dispatch guarantees low performance degradation even with extremely volatile peers. Compared with the non-redundant task dispatch, the redundant task dispatch can offer up to three times better performance. With the runtime optimization method that finds the near optimal number of redundant tasks, the computing platform can achieve a near optimal performance. In addition, the performance impact in a heterogeneous volunteer computing platform is evaluated. The simulation results indicate that the dependable workflow management mechanism works efficiently, even in a heterogeneous environment.

Second, a performance-oriented task dispatch policy is proposed and evaluated. It is an extension of the redundant task dispatch policy. The original redundant task dispatch policy shows a significant performance improvement over the non-redundant dispatch with

volatile peers in the evaluation. However, this policy has a major limitation: the average failure rate model is not the best fit for the volunteer peers in the real world. Thus, a new task dispatch policy is required to address this limitation. The proposed performance-oriented task dispatch policy is based on failure probability estimation. The existing prediction methods based on the resource availability status provide different accuracy for their selected environments (cluster, servers, PCs in corporate network, grid, and volunteer computing systems). Because this work requires only the failure probability, the empirical distribution of the *TTF* (Time-to-Fail) can provide enough information. Here, a heuristics-based mechanism for failure probability estimation is proposed based on the life cycle model of volunteer peers and the statistical *TTF* data. The runtime *TTF* data are gathered using a mechanism for periodically checking worker availability status. The performance-oriented task dispatch policy - *Least Failure Probability Dispatch (LFPD)* for volunteer computing platforms includes an enhanced workflow management mechanism that maintains runtime information such as the list of worker IDs that process the same task, the list of estimated failure probability for each copy of a task, and the estimated failure probability of this copy on a worker. It also includes a task selection policy and a task dispatch policy. The task with the highest failure probability is selected for dispatch when task enquiries come to the dispatcher. Because the failure probabilities of workers to which the same task is dispatched are different in a heterogeneous environment, the lower overall failure probability can be achieved by considering the task assignment of multiple tasks to multiple workers. Here, the idea of *dispatch window* is introduced. The dispatch window holds a number of tasks that will be dispatched together. Given a window size w, the dispatcher waits for task enquiries from workers until the dispatch window is full, then it selects w tasks with the *highest-failure-probability-selected* policy. The estimated failure probability is used to find the optimized task assignment that minimizes the overall failure probability of these tasks. This *LFPD* dispatch policy is evaluated with two real world trace data on a simulator. The evaluation results are compared with the results of two selected baseline policies. The comparison indicates the effectiveness of the *LFPD* policy. Furthermore, the results prove that the *LFPD* policy can beat the greedy dispatch policy when the mean task process time is much smaller than the mean *TTF* of the workers. The results also show that the *LFPD* policy is more efficient for a smaller team task process time, a larger number of task groups, a larger number of online workers. While multiple types of workers exist in the real world volunteer computing platforms, the *LFPD* is also evaluated with and without the ability to identify worker types, using a trace data with two types of workers. Results indicate that worker type identification can provide additional performance improvement.

At last, the potential of secure volunteer data processing using a proposed hardware-based cryptography approach is explored. Human beings are producing and gathering rapidly increasing of data (text, image, video etc) volume these years. Processing these data requires massive computing power. While the promising volunteer computing platforms have been used to process publicly available data, there are many other types of data processing that cannot utilize their massive computing power. To process the sensitive data such as medical data (patient data etc), biology data (DNA etc), market research data (customer data etc), financial data (bank account etc) on volunteer computing platforms, a feature for data security is required. This work explores the potential

of secure data processing on the volunteer computing platforms using a hardware-based cryptography method. The related work of privacy preserving data mining (*PPDM*) is reviewed. Some of the *PPDM* algorithms introduce side effects to the found knowledge, because the original data are modified. The cryptography-based *PPDM* algorithms have no side effects, but assume a different ownership of sensitive data. To enable secure volunteer data processing, a general cryptography-based secure data processing method for the volunteer computing is proposed. Before dispatching to the volunteer peers, plaintext data are encrypted by the data owner's server. The encrypted data are dispatched to the volunteer peers. The volunteer peers decrypt the data with the proper application key and process the plaintext data. By insuring the safety of the application key and denying the access to the plaintext data in memory, the sensitive data are protected on the volunteer peers. The decryption on the workers can be implemented using software-based and hardware-based approaches. The software-based approach trusts the main memory and stores the decrypted plaintext data in it. However, this approach is vulnerable because the operating system could be hijacked, the main memory is vulnerable to physical attacks, and the owners of volunteer peers can fully access their peers. The hardware-based approach that trusts neither the operating system nor the main memory is much more tamper resistant. The only trusted entity is a secure processor. The secure processors are designed to protect data from being exposed to unauthorized users because of the increasing demands for the avoidance of duplication and reverse-engineering of digital contents. The key hierarchy is protected through an embedded hardware key. The plaintext data only exist in the local memory of the secure processor. Thus, the data security is guaranteed even on a volunteer peer. A sample application and a widely available processor with hardware-based security features are used to study the hardware-based secure data processing method. Intensive optimizations for the architecture of the secure processor are applied to this application to archive high performance secure data processing. Evaluation results indicate that the secure version of a data processing application executed on secure hardware outperforms the non-secure version on commodity processors, even though the results also demonstrate a non-negligible performance overhead introduced by the security function.

Working together, these proposed solutions solve the issues on different layers of the existing volunteer computing platforms. Thus, a secure and high performance volunteer computing platform is achieved. It has the potential to solve a large number of complex computational problems on sensitive data, with the massive computing power provided by the million of volunteer peers.

Acknowledgements

I would like to thank all the people who have guided and supported me in accomplishing of this dissertation.

First, I would like to thank my supervisor Professor Hiroaki Kobayashi, for showing me both the patience and enthusiasm to supervise me during this past five years. His guidance and encourage have greatly helped me to accomplish this research. His careful review and helpful suggestion helped me to improve dissertation. I would also like to express my appreciation to Professor Susumu Horiguchi and Professor Hideaki Sone for taking on the responsibility of being on my dissertation committee, and their insightful comments.

I would like to express my sincerely gratitude to Associate Professor Hiroyuki Takizawa, for the enthusiastic discussions and precise suggestions throughout this research. I am also grateful for his valuable suggestion, and carefully proofreading my dissertation for errors in technical details, grammar and style.

I am also thankful to all the members of Ultra-Highspeed Information Processing Algorithm Laboratory, for their help and support. Associate Professor Hideaki Goto has provided many wise advices. I appreciate the thought-provoking discussions I have enjoyed with Assistant Processor Ryusuke Egawa on the seminars. Yoshitomo Murata has helped me to set up the evaluation environments. Seiji Saitoh and Yoshikuni Kataoka who graduated several years ago, have helped me to settle down in Japan.

I am also grateful to Ministry of Education, Culture, Sports, Science and Technology of Japan, for the financial support.

Last but not least, I would like to thank my parents, my girlfriend and my friends for their encouragement toward the completion of this research.

Hong Wang

Janurary 2009

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The aim of volunteer computing [1] (also known as “P2P Computing” or “Public-resource Computing”) is to use the Internet-connected individual computers to solve computing problems, especially large-scale scientific computational problems. In mid-1990s, there are two pioneering research projects, including GIMPS [2] (The Great Internet Mersenne Prime Search) and Distributed.net [3]. GIMPS is a distributed computing project researching Mersenne prime numbers. Distributed.net is a general purpose computing platform. In 1999, SETI@home [4, 5] was launched by the Space Sciences Laboratory, at the University of California, Berkeley. The purpose of SETI@home is to analyze data incoming from the Arecibo radio telescope, searching for a possible evidence of radio transmissions from extraterrestrial intelligence.

These projects are rather successful. GIMPS has already found a total of 9 Mersenne primes, each of which was the largest known prime number at the time of discovery. While SETI@home has not found any conclusive signs of extraterrestrial intelligence, it has identified several candidate spots for further analysis. Distributed.net has successfully provides the solutions of the DES, RC5-32/12/7 (”RC5-56”), and RC5-32/12/8 (”RC5-64”) of the RSA secret-key challenge.

Nowadays, the world's computing power is distributed in hundreds of millions of personal computers, game consoles and other computing devices. Using the idle cycles from Internet-connected PCs and game consoles, the volunteer computing platforms have the potential to provide more computing power than any supercomputers, clusters, or grid, and the disparity will grow over time.

1.2 Volunteer Computing Platforms

There are several well-known volunteer computing platforms such as Folding@home [6], BOINC [7] (Berkeley Open Infrastructure for Network Computing), XtremWeb [8], Entropia [9], Alchemi [10], and JNGI [11] to name a few.

Folding@home [6] is a volunteer computing platform designed to study protein folding, misfolding, aggregation, and related diseases. It works on heterogeneous computers running Windows, Mac, variants of UNIX-like operating systems, and PlayStation3. It utilizes the computing power of not only general purpose CPUs, but also a graphics processing unit (GPU) and the Cell processor inside PlayStation3. Currently, it is the most powerful volunteer computing platform in the world.

BOINC [7] harnesses hundreds and thousands of PCs across the Internet to process a massive amount of data of different scientific computing problems, including radio telescope data analysis, proteins shapes analysis, searching for spinning neutron stars. Similar to Folding@home, it works on heterogeneous computers running Windows, Mac, and variants of UNIX-like operating systems.

XtremWeb [8] consists of three kinds of peers: the coordinator, the workers and the clients. Compared to some other platforms, it allows multiple coordinators and clients (workers keep a collection of IP addresses of coordinators and clients).

Entropia [9] is a Windows volunteer computing system for enterprise networks. It differentiates the others for the sandboxed task execution. Binary modification is used to intercept all important Windows API calls.

Alchemi [10] is a .NET-based framework for constructing desktop grids and developing grid applications. It supports an object-oriented application programming model in addition to a file-based job model. Cross-platform support is provided via web services.

JNGI [11] is a volunteer computing platform based on JXTA [12, 13]. The platform utilizes the JXTA peer-to-peer communication protocol [14] to construct the overlay network. To improve scalability, it organizes computational resources into groups through the support of JXTA peer groups. Thus, the top level monitor will not be the bottleneck.

By December 2008, the most powerful volunteer computing platform - Folding@home [6] achieved more than four Petaflops computing power by connecting more than 3,500,000 CPUs [15]. Among the volunteer nodes, more than 600,000 PlayStation3 contribute about 1.7 Petaflops computing power. The second powerful volunteer computing platform - BOINC provided a sustained processing power of more than one Petaflops [7] since Januray 2008. In contrast, the fastest supercomputer, Roadrunner achieves a maximum LINPACK performance of 1.026 Petaflops [16].

The General Architecture of the Existing Volunteer Computing Platforms

The general architecture of the existing volunteer computing platforms is shown in Figure 1.1. Using the master-worker model, it usually consists of two kinds of peers: dispatchers and workers. For most existing platforms, the dispatcher is a specified server. The workers are volatile peers of the volunteer platform.

Here, JNGI is used as an example volunteer platform. A computational job of JNGI consists of independent tasks of the embarrassingly parallel computation model. A task is a Java program with its own data. Tasks are distributed among worker peers to process. A computational job is processed by JNGI in the following four steps.

1. A job submitter submits a computational job to a monitor group. A monitor group is a set of peers that works as a portal server of JNGI.

2. A task dispatcher (the name of the master peer in JNGI) dispatches independent

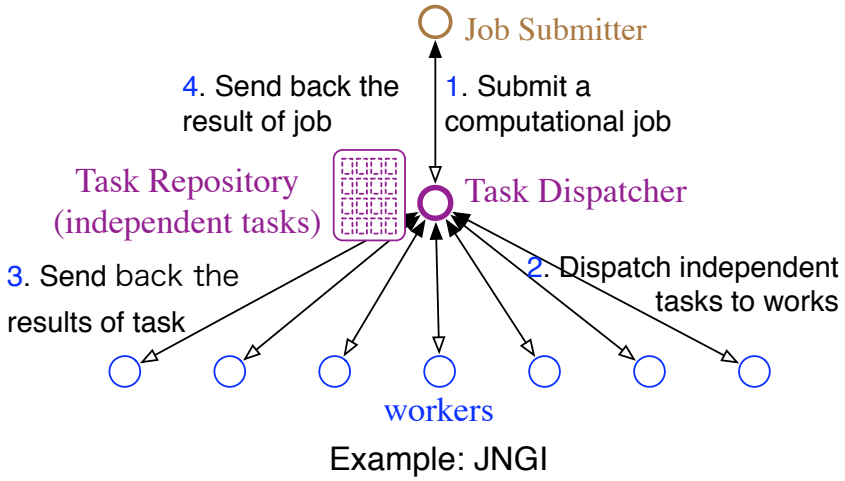


Figure 1.1: *General architecture of the volunteer computing platforms.*

tasks of the job to worker peers.

3. A worker peer sends back the result of the task to the master peer after the task is completed.

4. While all results of tasks in the job are received, the dispatcher peer sends back the result of the job to the job submitter.

The Two Issues with the Existing Volunteer Computing Platforms

Despite the massive computing power offered by the existing volunteer computing platforms, there are two major issues with them that limit their applicable areas.

The existing volunteer computing platforms are lacking support for inter-task dependency. Thus, they can only solve embarrassingly parallel problems [17] that can obviously be divided into a number of completely independent tasks. However, it is only minority of all the computational problems. To make the volunteer computing platforms more applicable for general uses, computational problems with inter-task dependency have to be supported. However, inter-task dependency results in a status that none of the un-dispatched tasks can be dispatched, because these un-dispatched tasks require the results of one or several of the tasks that are being executed. This status may lead to serious per-

formance degradation, because of the frequent task failures of volatile peers in volunteer computing platforms. Therefore, a mechanism that makes use of idle peers to guarantee low performance degradation for task failures is required.

The other issue is that the volunteer computing cannot be applied to the sensitive data processing, because of privacy and security concerns. Thus, volunteer computing has not been applied to the application fields other than research oriented computation on publicly available data. To apply volunteer computing to privacy sensitive data processing, security features are strongly desired on the volunteer computing platforms.

1.3 Objective of the Dissertation

The objective of this research is to establish a widely applicable volunteer computing platform, focusing on solving the two major issues with the existing volunteer computing platforms. Figure 1.2 shows the issues and their corresponding solutions on different layers and components (dispatcher and worker) of the proposed volunteer computing platform. The task management layer is responsible for the management of tasks in a computational job. It decides the task to dispatch, and maintains the information of tasks at runtime. The resource management layer consists of the functions to gather the runtime resource information, and decides the resource to dispatch. The application layer handles the application specific problems.

On the task management layer, the existing volunteer computing platforms lack of support for inter-task dependency. A solution is to introduce workflow management mechanism to the task management layer. Because the serious performance degradation is introduced by the inter-task dependency on volunteer computing platforms, optimized task dispatch policies are required on the resource management layer to mitigate the performance degradation. To implement the optimized task dispatch policies, worker availability checking is also added to the resource management layer to gather runtime information. This availability checking involves both the dispatcher and the workers. Two

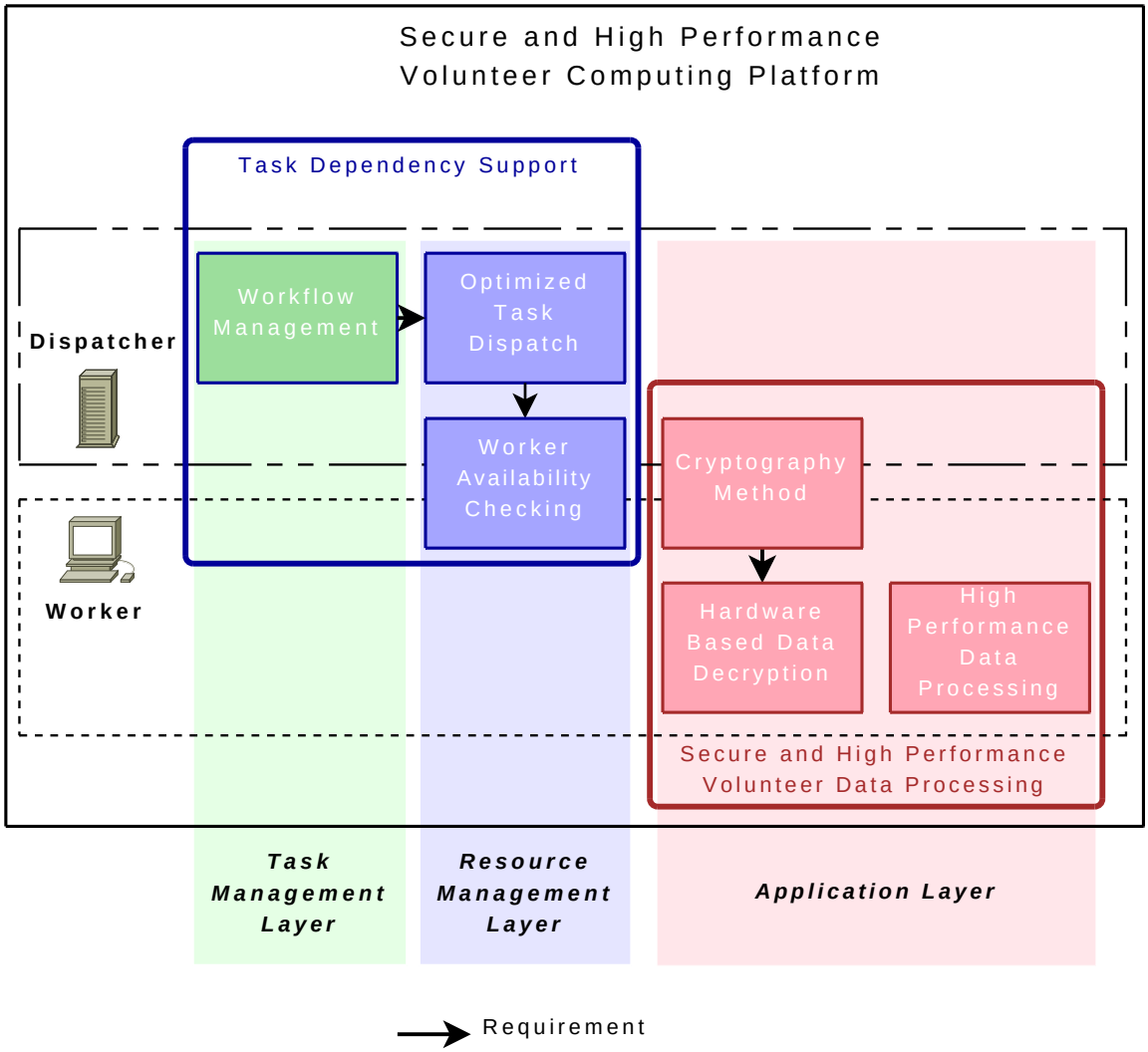


Figure 1.2: *The overview of secure and high performance volunteer computing platform.*

task dispatch policies are proposed to find the optimized task dispatch decisions based on this information.

On the application layer, the exiting volunteer computing platforms are not equipped with security features to protect sensitive data on the workers. A proposed solution is a cryptography-based secure data processing method that involves both the dispatcher and the workers. Sensitive data is encrypted on the data owner's server and sent to the worker. Then the workers decrypt the data and process it. While the encryption of the data owner's server can be trusted, the decryption of the volunteer workers is challenging. The decryption can be implemented in a software-based approach or a hardware-based approach. The software-based approach is vulnerable for the volunteer computing. The hardware-based approach using secure processors is more tamper-resistant. The potential of the hardware-based approach is studied using a sample application on a widely available secure processor in the volunteer computing platforms. The application is intensively optimized for the architecture of this secure processor to archive high performance secure data processing.

1.4 Organization of the Dissertation

The rest of the dissertation is organized as follows. Chapter 2 proposes a dependable workflow management mechanism for the volunteer computing platforms. A redundant task dispatch policy is designed and evaluated. Chapter 3 presents a performance-oriented task dispatch policy and evaluates its efficiency with real world trace data. Chapter 4 explores the potential of secure data processing on the volunteer computing systems. It presents a hardware-based secure data processing method, and studies the performance of an example application of this method on a secure processor. Chapter 5 concludes and summarizes this dissertation.

Chapter 2

A Dependable Workflow Management Mechanism for Volunteer Computing

2.1 Introduction

In this chapter, a dependable workflow management mechanism is presented. The workflow management is introduced to volunteer computing platforms to extend the applicable areas. The dispatcher also needs to consider the overheads caused by task failures, because the tasks must wait for the preceding tasks' results if a volunteer computing system with volatile peers supports inter-task dependency. Therefore, a redundant task dispatch mechanism is proposed to reduce the overheads. Redundant task dispatch uses the computing power from the idle workers to process duplicate copies of tasks, thus the task failure probability can be reduced. Therefore, the performance degradation can be reduced. However, unlimited redundant task dispatch may result in performance degradation, because too much computing power is wasted for the processing of duplicate copies. As the optimal value for the number of redundant tasks depends on the runtime parameters that are unknown in a real computing environment, a runtime optimization is designed for

the redundant task dispatch to find the optimal value. The performance of the proposed mechanisms is evaluated using a simulator.

2.2 Related Work

No workflow management mechanism exists for volunteer computing platforms. This is because of the fact that volunteer computing platforms are mostly used for solving embarrassingly parallel problems and not for other kinds of computational problems. Workflow management mechanisms have been studied a lot for grid systems.

2.2.1 P2P-RPC

P2P-RPC [18] is a simple programming interface to develop applications on XtremWeb [8]. It is implemented as a remote procedure call (RPC) API that is derived from OmniRPC [19] (an existing RPC API for the grid based on Ninf [20] system). This new API is implemented on top of the low-level functionality of the XtremWeb volunteer computing system.

The concept of RPC has been used for a long time in distributed computing as it provides a simple way for communications among distributed components. With asynchronous and synchronous calls of P2P-RPC, a programmer can define tasks and a process sequence of these tasks.

For the RPC mechanism, the procedures need to be installed on worker peers in advance. Since a volunteer computing platform is a dynamic system that consists of more than thousands of peers, this requirement makes it inconvenient to solve different computational problems.

Fault tolerance is insured by the underlying environment (XtremWeb). A failed task will be dispatched again to another worker after the failure is detected by a time-out mechanism. With such a fault tolerant mechanism, any worker's failure will not affect the execution (but the performance). The performance degradation caused by worker's

failure is not considered in XtremWeb.

2.2.2 Grid Workflow Management

In the field of grid, workflow management mechanisms have been studied a lot. Yu et al. [21] proposed a taxonomy that characterizes and classifies various approaches for grid workflow. Some of the related work are listed:

- WebFlow [22] is a pioneering work, which is a visual programming paradigm to develop high performance distributed computing applications.
- Bivens has defined a grid workflow specification [23] in XML, and used in the ASCI (Accelerated Strategic Computing Initiative) grid infrastructure.
- GridFlow [24] is a two-layer workflow management system for grid computing, including global grid job workflow management and local grid sub-workflow scheduling.
- GridAnt [25] extends the vocabulary of Apache Ant [26] to support workflow management in grid.
- The CCA(Common Component Architecture) [27] and its XML implementation have been developed for grid programming.
- Symphony [28] is a framework for combining existing codes to meta-programs without changing the code.
- CXML(Component XML) [29] is used for component specification and further issues such as performance optimization.
- Neubauer et al. [30] implemented a simple, Petri net-based graph model for grid services on OGSI-compatible grid middlewares.

- Grid-WFS [31] is a flexible framework for fault tolerance in the grid. Replication in Grid-WFS lets a user to specify a particular task to be replicated on multiple grid resources.

- Abawajy [32] proposed a fault-tolerant scheduling policy for grid systems with a job replication mechanism. This policy assumes that a grid environment is underutilized and thus spare resources can be used for replication job execution without affecting the overall performance.

The key issue that differentiates this work from the related studies on grid is that optimized scheduling/mapping of tasks to resources is always considered for workflow management in grid computing. Because the volunteer computing platforms consist of much more volatile peers, the performance degradation caused by task failures needs to be considered first, rather than the optimal scheduling of tasks.

Grid-WFS [31] provides a user defined task replication scheme. However, this user defined approach is not suitable for the volunteer computing platforms, because user cannot specify thousands of tasks to anonymous peers. In most volunteer computing platforms, worker peers are highly utilized after they joined the computing platforms. Therefore, the scheduling policy proposed in [32] will result in performance degradation, because it is designed for the environment that is highly underutilized.

2.2.3 Runtime Task Replication Management

Litke et al. [33] have designed a task replication scheme for grid environments. The objective of their research is to maximize the grid resource utilization and the achieved profit. The task replication management mechanism can find a proper number of replicas to maximize the profit of grid resources, while fulfilling the deadline constraint of a computing job defined by an end user. Since worker peers are volunteers in the volunteer computing platforms, the profit is not considered while finding the optimal number of replicas. Moreover, task dependency is not discussed in their work.

2.2.4 Research Issues

Since applicable areas of the existing volunteer computing platforms are limited to embarrassingly parallel problems, a volunteer computing platform that can solve more kinds of computing problems is required. The requirement leads to two research issues: task dependency and dependability.

Task Dependency

To solve a computational problem with task dependency on existing volunteer computing platforms, the dependency has to be handled on the job submitter side. The only way is to divide the computational problem into several groups of tasks with dependency among them. Each task group consists of independent tasks. A program on the job submitter side handles the dependency. A group of tasks is sent to the volunteer computing platform as a job, so that it just solves an embarrassingly parallel problem. After the volunteer computing platform finished the processing of one group, the results of this task group are sent back to the job submitter. Then the program on the job submitter gives the results of the finished task group to another task group. When a task group has all its required data, the program on the job submitter will submit it to the volunteer computing platform to process. In this way, P2P-RPC [18] extends the applicable areas by defining the task process sequence with asynchronous and synchronous calls.

In such a process of computational problems with task dependency, the job submitter has to submit jobs and to receive their results for lots of times. It results in two problems. First, since the program on the job submitter controls the entire process sequence, this computation cannot continue whenever the job submitter crashes or other problems happen. If the task dependency can be handled automatically by volunteer computing platforms, the job submitter just needs to submit a job and wait for its results. Even it crashes at the runtime, it can still receive the results from the volunteer computing platform after its restart. Second, compared with submitting all the tasks at one time, the

communication cost of multiple job submissions and result receptions is likely to be much higher. Therefore, it is not suitable for large scale, highly volatile volunteer computing platforms. A workflow management mechanism to control the job's process sequence is required for a widely applicable volunteer computing platform.

Dependability

Grid workflow management has been well studied. However, because of the dynamic nature of volunteer networks, none of the grid workflow management mechanism is suitable for volunteer computing. This is because of the lack of functionality to cover the low-availability of volatile peers. Therefore, high dependability is strongly required for volunteer computing platforms in order to make them more universal as high performance computing platforms. In this dissertation, the *dependability* is defined by Equation (2.1) to represent the performance effects of task failures:

$$Dependability = \frac{Performance_{actual}}{Performance_{no\ failure}}, \tag{2.1}$$

where $Performance_{no\ failure}$ denotes the peak performance without task failures, and $Performance_{actual}$ is the actual performance with volatile peers. In this chapter, Equation (2.1) is used as the objective function to improve the dependability of a volunteer computing platform.

2.3 Dependable Workflow Management Mechanism

The dependable workflow management mechanism proposed in this chapter has the following three key features.

- A workflow management mechanism is designed to extend the applicable areas of computing problems by supporting task dependency.
- Redundant task dispatch is designed to guarantee a high dependability.

- A runtime optimization method based on mathematical analysis is proposed to achieve a near optimal dependability.

Classified with the taxonomy of grid workflow management [21], this mechanism features a non-DAG, abstract, user-defined workflow. For scheduling, it uses a centralized dynamic performance-driven policy.

2.3.1 Basic Idea

A workflow management mechanism analyzes the status of tasks to decide whether a task can be dispatched or not. Task dependency results in a kind of status, in which none of the tasks can be dispatched because of the dependency. In such status, worker peers will be idle and keep waiting for a new task for a long time. In addition, because a volunteer computing platform consists of volatile peers, its high task failure rate makes this problem worse.

To simplify the discussion, it is assumed that the volunteer computing platform to be a homogeneous environment that all the workers have the same performance and task failure probability does not change over time. This dissertation focuses on the computation-intensive problems that "*computation time* $\gg$ *data transfer time*." Therefore, the communication cost is not discussed.

The execution time for the set of independent tasks without the redundant task dispatch is shown in Equation (2.2).

$$Time_{no\ redundant} = \left\{ \sum_{i=0}^{\lfloor \log_{FR} \frac{1}{N} \rfloor} \left\lceil \frac{N \times FR^i}{W} \right\rceil + \left\lceil \sum_{i=\lfloor \log_{FR} \frac{1}{N} \rfloor + 1}^{+\infty} N \times FR^i \right\rceil \right\} \times t, \quad (2.2)$$

where $FR(0 < FR < 1)$ denotes the constant task failure rate, N denotes the number of independent tasks to be processed, t is the process time of each task, W is the number of workers, and i is the number of consecutive failures occuring for one task. Execution time increases with the task failure rate. While $N \times FR^i < W$, parts of the workers are idle.

While $N \times FR^i < 1$, the probability that a failure occurs is used to count the process time.

As redundant task dispatch uses the computing power from these idle workers to process duplicate copies of tasks, it can reduce the task failure rate and thus increase the possibility of reducing the performance degradation. On the other hand, unlimited redundant task dispatch may result in performance degradation because too much computing power is wasted for processing extra copies of tasks. Therefore, an optimization method to limit the redundant task dispatch to a proper level is required. While the redundant task dispatch exceeds a proper level, the workflow management mechanism stops using the redundant task dispatch, and the idle workers are reserved for the process of upcoming tasks. Working together, these three features provide a widely applicable, highly dependable volunteer computing platform.

2.3.2 Workflow Management Mechanism

A dependable workflow management mechanism for the volunteer computing platform has the following basic requirements:

Workflow Markup Language: A programming interface for programmers to define a workflow.

Workflow Management Module: A software component that maintains the task dependency and redundancy information. It also controls the process sequence with this information. A runtime optimization method helps it to limit the usage of redundant task dispatch to a proper level.

For the programming interface, a vocabulary of the workflow markup language is defined in XML. It provides a simple interface to define a workflow of executing tasks in a computational job. A workflow description file in the markup language is parsed into a workflow graph, which can be used by the workflow management module. The

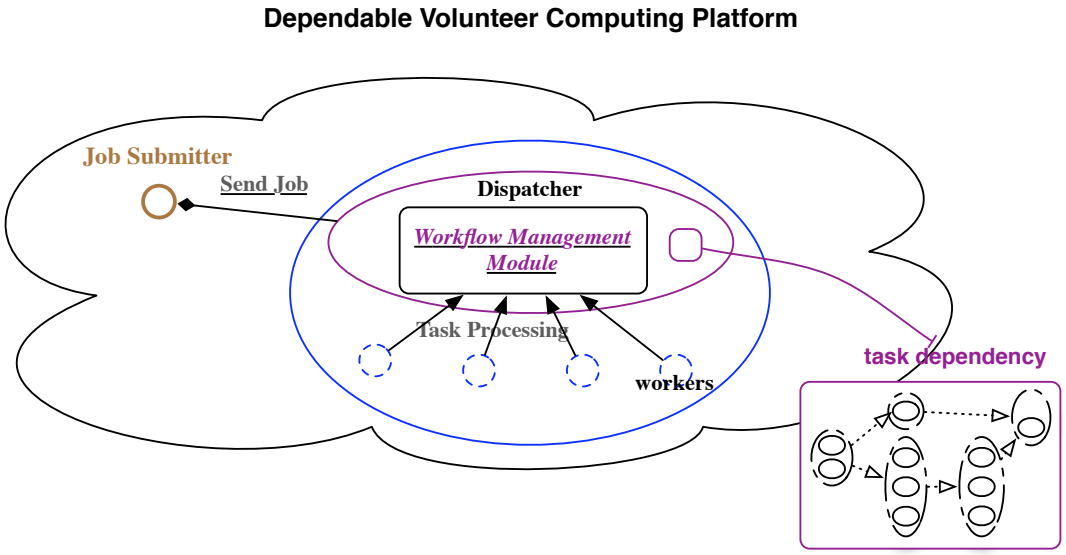


Figure 2.1: *Overview of the dependable workflow management mechanism for volunteer computing platforms.*

parsed workflow graph has all the dependency information. Redundancy information is initialized and maintained at the runtime.

As shown in Figure 2.1, a volunteer computing platform with the the dependable workflow management mechanism consists of two kinds of peers: workers, and dispatchers. The workflow management module is developed as a component of the dispatcher.

The process flow of a computing job with the mechanism is as follows:

1. A job (a set of tasks and the description of their workflow) is submitted from a job submitter peer to the dispatcher.

2. On the dispatcher, a workflow description file is parsed and translated into a workflow graph. The workflow management module on the dispatcher maintains information in the workflow graph, and controls the task process sequence with a redundant task dispatch and a runtime optimization method.

Workflow Markup Language

This workflow management mechanism works with a set of predefined tasks. To define the tasks, a workflow markup language is presented in XML. This is because of the following

reasons:

- An open standard with the W3C consortium and standards committees [34].
- Platform independent: suitable for the heterogeneous environment of volunteer computing platforms.
- Human readable: make it easy for programmers to define a workflow.

The workflow elements currently used are defined as follows:

- *Workflow*: the "root" element of a workflow definition. It is a container for tasks to be executed by the workflow management module.
- *Group*: an element of a task group. A task group is a set of tasks that belong to the same program, and can be executed independently. Each group has an attribute *id*, which is a unique identifier of the task group in the current job.
- *TaskList:* the list of tasks in a task group. *TaskList* has two attributes: *first*(identifier of the first task in this task group) and *last*(identifier of the last task in this task group). Each of them is the unique identifier of a task in the current job. They are used to define which tasks are included in this task group.
- *Description*: the description of the task group. A programmer can use this element to write some comments.
- *Dependency*: the dependency of a task group. It has two attributes: *groupID* and *arg*. Attribute *groupID* defines which task group this one depends on. Attribute *arg* (the ID of a variable) defines which variable of the tasks in this task group needs to be updated. A task group can have several *Dependency* elements.
- *Loop*: a loop body. It has an attribute - *loop_number* that defines how many times the loop will be processed.

- *Replace*: the dependency between loop initial parameters and the results after one iteration. Its attribute is the same as *Dependency*.

Here, an example of the workflow description file is given.

```
<Workflow>
  <Group id=1>
    <TaskList first="1" last="2" />
    <Description>Test Task Group 1
    </Description>
  </Group>
  <Loop loop_number=100>
    <Group id=2>
      <TaskList first="3" last="5" />
      <Dependency groupID="1" arg="2"/>
      <Replace groupID="3" arg="3"/>
    </Group>
    <Group id=3>
      <TaskList first="6" last="8" />
      <Dependency groupID="2" arg="1" />
    </Group>
  </Loop>
  <Group id=4>
    <TaskList first="9" last="10" />
    <Dependency groupID="1" arg="1" />
    <Dependency groupID="3" arg="2" />
  </Group>
</Workflow>
```

The workflow graph of the example workflow description file is shown in Figure 2.2. Task Group 1 consists of two tasks: Tasks 1 and 2. The result of Task Group 1 is required to execute Task Groups 2 and 4. Task Group 3 uses the result of Task Group 2 as the first input parameter, and Task Group 4 needs the results of Task Groups 1 and 3. Task Groups 2 and 3 are in the loop body. The loop will be processed 100 times. After one

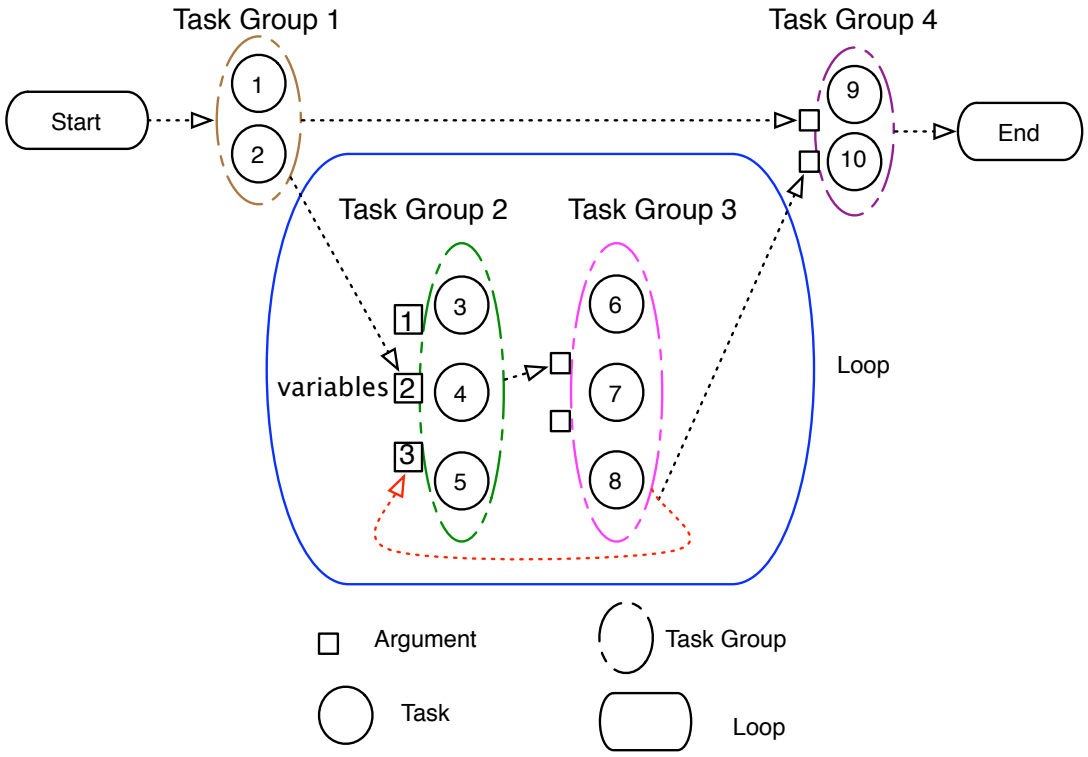


Figure 2.2: *Example of a workflow graph.*

iteration, the result of Task Group 3 will be used to replace the third input parameter of Task Group 2.

Workflow Management Module

The workflow management module is a component of the dispatcher. The module is responsible for directing the workflow control and the task information update. After a job is dispatched to a worker group, its workflow description file is parsed and translated into a workflow graph by a workflow parser. In a workflow graph, each task has its own status and redundancy rate. The redundancy rate (RR) is used to control redundant task dispatch and record how many workers process the task at the same time. Working with the workflow management module, redundant dispatch is designed to handle the dynamic nature of volunteer computing platforms by dispatching redundant tasks. The redundant task dispatch with an appropriate RR can improve the performance.

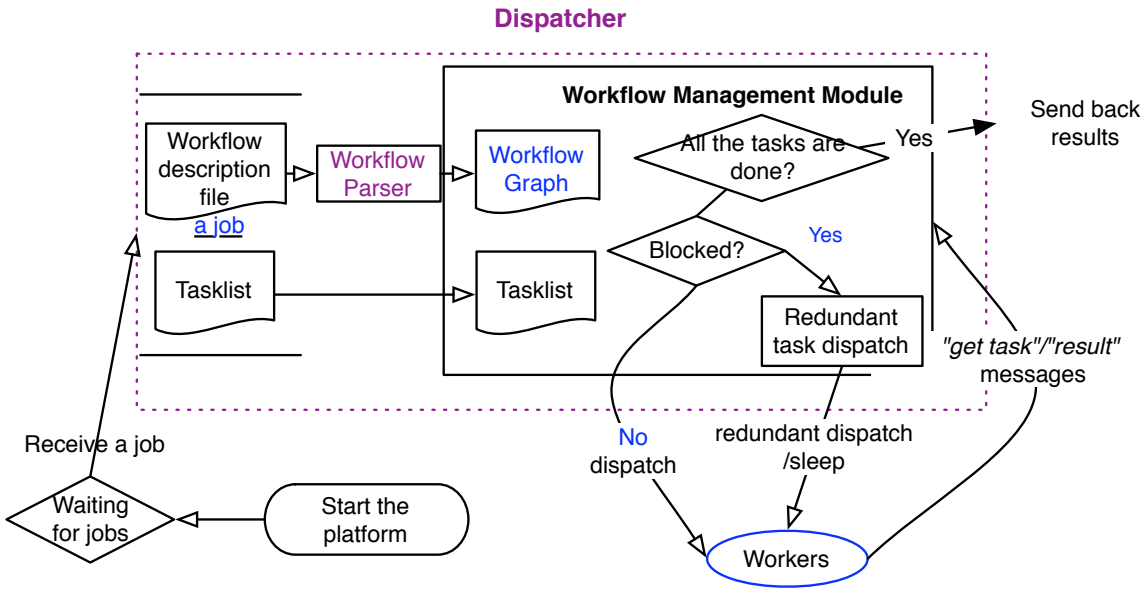


Figure 2.3: *Process diagram of the dispatcher.*

The process diagram of the workflow management module is shown in Figure 2.3. There are three kinds of status of each task: "undispatched", "dispatched", and "finished." The initial status of each task is "undispatched" and the initial RR is 0. Worker peers inquire the dispatcher for new tasks when they are idle or finish their tasks. When the workflow management module dispatches an "undispatched" task, it provides the required input values from the finished tasks, and then changes this task's status to "dispatched" and increases RR by one. When a task result is received by the workflow management module, the status of this task is set to "finished." When a task's result is not received after a predefined period, the task is assumed to be failed and RR of this task is decreased by one. The workflow management module uses the status information to analyze whether an "undispatched" task can be dispatched or not. An "undispatched" task can only be dispatched when all the tasks that it depends on are "finished." A workflow has two kinds of status: "blocked" and "unblocked." While there is no such an "undispatched" task, the workflow management module uses the redundant task dispatch to achieve a low performance degradation. Such status of a workflow is defined as "blocked."

2.3.3 Redundant Task Dispatch

To guarantee a low failure rate of the "dispatched" tasks, redundant task dispatch selects a "dispatched" task with the least RR and dispatch this task to an idle worker. This selection policy is named *least-RR-selected*. The *least-RR-selected* policy equally reduces the failure rate of all the "dispatched" tasks. If the failure rate is not reduced equally, some of the tasks with lower RR still have a great chance to encounter a failure. Any failure of the "dispatched" tasks will lead to re-execution, and therefore will result in performance degradation.

Suppose task "a" is dispatched to worker "A," the workflow status is "blocked" because the result of task "a" is required, and another worker "B" inquires for a new task after the workflow status becomes "blocked." Then, as shown in Figure 2.4, three cases can be discussed as follows.

Case 1 Worker "A" successfully finishes the processing of Task "a": the result of Task "a" will be returned at time point "Z."

Case 2 Worker "A" encounters a failure; no redundant task dispatch: Task "a" will not be dispatched again until the failure is noticed. Worker "B" periodically inquires for a new task and always get a "sleep" reply from the dispatcher. After the failure of Worker "A" is noticed, Worker "B" gets Task "a." Without any failure, Worker "B" returns the result of Task "a" at time point "Y."

Case 3 Worker "A" encounters a failure; with redundant task dispatch: there might be two or more workers processing the same task. Here, only two workers are considered. Worker "B" gets a duplicate Task "a" even when the workflow is "blocked." Even if Worker "A" encounters a failure, Worker "B" can return the result of Task "a" at time point "X," which is much earlier.

The relation of these three time points can be described as $Z \leq X < Y$. If Workers "A" and "B" get Task "a" at the same time, $Z = X < Y$. If Worker "B" gets the Task

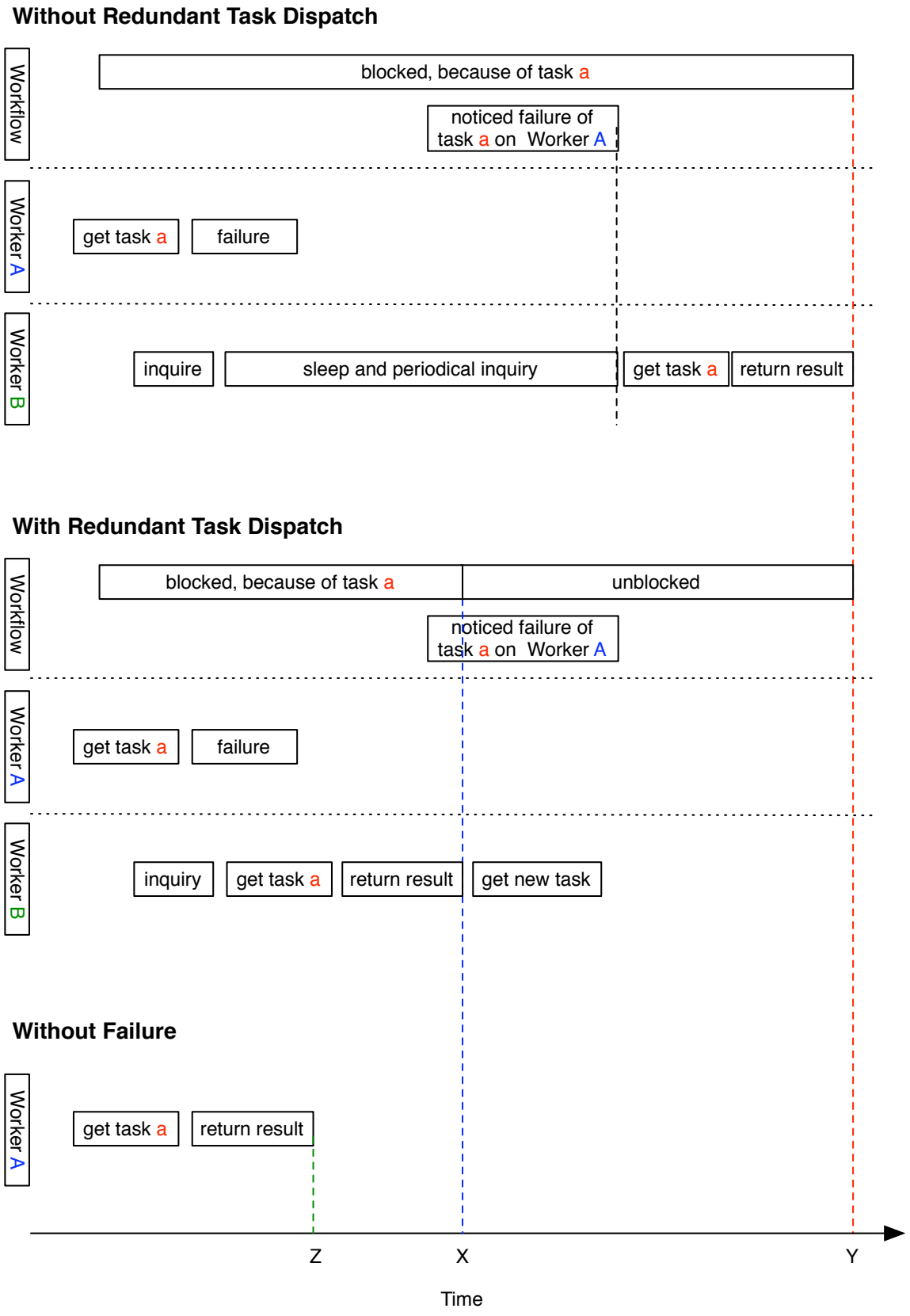


Figure 2.4: *Comparison of the execution time for a example of workflow status.*

"a" later than Worker "A", $Z < X < Y$. With redundant task dispatch, the result is returned at time point "Y" only if both of these two workers encounter failures. These three cases show that redundant task dispatch can reduce the performance degradation to a much lower level with volatile peers.

The re-execution of failed tasks results in the serious performance degradation in Case 2. With redundant task dispatch, task re-execution in Case 2 is required only if all the workers processing the same task encounter failures. Suppose the current average failure rate of each worker is AFR ($0 \leq AFR < 1$), then the expected failure rate of a task that is processed by x workers is AFR^x. This expectation is much lower compared to the non-redundant one. Therefore, redundant task dispatch can save lots of time from re-execution of failed tasks with frequent peer failures. Thus, the performance degradation caused by volatile peers can be reduced.

2.3.4 Acceptable Redundancy Rate

Redundant task dispatch can avoid the performance degradation caused by re-execution of failed tasks. On the other hand, unlimited redundant task dispatch may result in performance degradation because too much computing power is wasted for processing duplicate copies of tasks. In this case, when the workflow is no longer "blocked," it is possible that there are many available tasks while few workers can start processing in a short time. This situation may also result in performance degradation. To have a trade-off between performance degradation caused by task failure and performance degradation caused by redundant task dispatch, ARR is defined as the acceptable redundancy rate for one task. The RR value ($RR \leq ARR$) of a task records how many workers are processing this task at the same time.

Redundant Dispatch Policy

When the workflow management module finds a "blocked" status of a workflow, redundant task dispatch is activated. The "dispatched" task list is generated. Here the *least-RR-selected* policy is used to select a task from the incomplete task list. If the RR of this task is less than ARR, the task is dispatched and the RR is increased by one. If the RR of this task is equal to ARR, only a "sleep" message is sent to the idle worker. An idle worker that received a "sleep" message waits a certain interval, and then inquires the dispatcher for a new task.

Execution Time with ARR

Let ${N_{incomplete}}^i$ be the number of incomplete tasks after the i-th dispatch. In a heterogeneous environment in which individual peers have difference performance, the AFR of different tasks varies, and may change over time. These real world characteristics lead to random parameters in the mathematical model. Since the AFR at each dispatch time point varies, this modeling is extremely difficult, especially while the distribution of task process times on different workers is unknown. To simplify the modeling, a homogeneous environment with constant FR in the mathematical model is assumed. The execution can be analyzed as the following three stages:

1. While ${N_{incomplete}}^i \geq W$: The number of workers is smaller than the number of incomplete tasks that can be calculated with Equation (2.3). No duplicate tasks will be dispatched. Incomplete tasks are dispatched once in the $(i+1)$-th dispatch. The execution time of this stage is shown in Equation (2.4).

$${N_{incomplete}}^i = N \times FR^i \ (0 \leq i \leq \left\lfloor \log_{FR} \frac{W}{N} \right\rfloor). \tag{2.3}$$

$$Time_{redundant\ 1} = \sum_{i=O}^{\left\lfloor \log_{FR} \frac{W}{N} \right\rfloor} \left\lceil \frac{{N_{incomplete}}^i}{W} \right\rceil \times t. \tag{2.4}$$

2. While $\frac{W}{ARR} < {N_{incomplete}}^{i} < W$: Duplicate copies of tasks will be dispatched. However, there are not enough workers to dispatch ARR duplications for all the incomplete tasks. Based on the redundant dispatch policy, the maximum possible RR is:

$$ {RR_{max}}^{i} = \left\lceil \frac{W}{{N_{incomplete}}^{i}} \right\rceil . \tag{2.5} $$

Let ${N_\alpha}^{i}$ be the number of task that RR_{max} duplications will be dispatched, and ${N_\beta}^{i}$ be the number of task that $RR_{max} - 1$ duplications will be dispatched. Then these two parameters can be calculated in Equation (2.6).

$$ \begin{cases} {N_\alpha}^{i} = W - {RR_{max}}^{i} \times {N_{incomplete}}^{i}. \\ {N_\beta}^{i} = (1 + {RR_{max}}^{i}) \times {N_{incomplete}}^{i} - W. \end{cases} \tag{2.6} $$

Let i^δ satisfy the following conditions:

$$ \begin{cases} {N_{incomplete}}^{i^\delta + 1} \leq \frac{W}{ARR}. \\ {N_{incomplete}}^{i^\delta} > \frac{W}{ARR}. \end{cases} \tag{2.7} $$

Thus, this stage's $i \in \left[\left\lfloor \log_{FR} \frac{W}{N} \right\rfloor + 1, i^\delta \right]$. The number of incomplete tasks is computed recursively as in Equation (2.8):

$$ \begin{cases} {N_{incomplete}}^{i} = {N_\alpha}^{i-1} \times FR^{{RR_{max}}^{i-1}} + {N_\beta}^{i-1} \times FR^{{RR_{max}}^{i-1} - 1}. \\ {N_{incomplete}}^{\left\lfloor \log_{FR} \frac{W}{N} \right\rfloor + 1} = N \times FR^{\left\lfloor \log_{FR} \frac{W}{N} \right\rfloor + 1}. \end{cases} \tag{2.8} $$

Finally, the execution time of this stage is shown in Equation (2.9).

$$ Time_{redundant\ 2} = (i^\delta - \left\lfloor \log_{FR} \frac{W}{N} \right\rfloor) \times t. \tag{2.9} $$

3. While $\frac{W}{ARR} \geq {N_{incomplete}}^{i}$: Duplicate copies of tasks will be dispatched. ARR duplications will be dispatched for all the incomplete tasks in the $(i+1)$-th dispatch. The number of incomplete tasks in this stage can be calculated recursively in Equation (2.10).

$$ {N_{incomplete}}^{i} = {N_{incomplete}}^{i-1} \times FR^{ARR} \ (i \in \left[i^{\delta}, +\infty\right)). \tag{2.10} $$

Each dispatch round takes time t. While ${N_{incomplete}}^{i} < 1$, the probability that a failure occurs is used to calculate the execution time. The execution time of this stage is shown in Equation (2.11).

$$ Time_{redundant\ 3} = \left\lceil \sum_{i=i^{\delta}}^{+\infty} \frac{{N_{incomplete}}^{i}}{\lceil {N_{incomplete}}^{i} \rceil} \right\rceil \times t. \tag{2.11} $$

2.3.5 Runtime Optimization

ARR is proposed to have a trade-off between performance degradation for task failure and performance degradation for redundant task dispatch. The value of ARR has a great effect on the total execution time. If ARR is too small, with extremely volatile peers, task failures happen frequently. The failed tasks require re-execution. Therefore, more task failures can possibly lead to a longer total execution time. If ARR is too big, while the task failure rarely happens, it results in performance degradation for wasting too much computing power for redundant task process. To appropriately adjust ARR, a runtime optimization method for ARR is required.

The previous analysis in Section 2.3.4 is based on the assumption that FR is constant value. While the FR may change over time, a proper value of ARR cannot be deducted from the execution time equations. An optimization method is proposed to find a proper value of ARR based on the analysis of relations among $N_{incomplete}$ (*the number of incomplete tasks that can be dispatched*), W (*the number of workers*), and AFR (*current average*

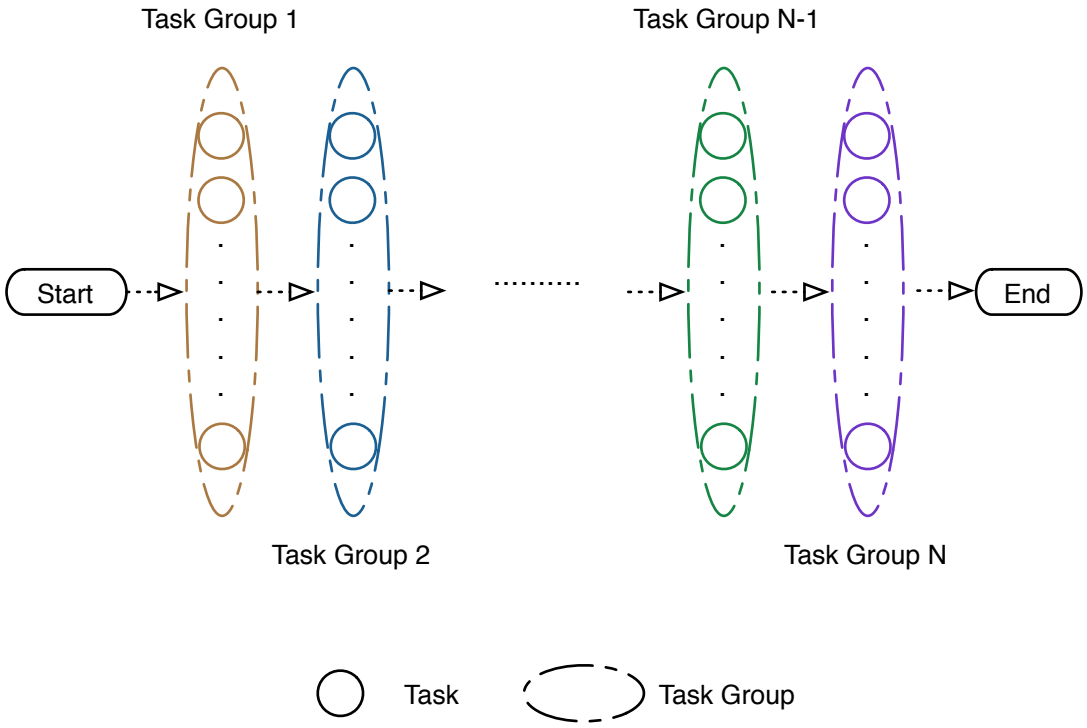


Figure 2.5: *The simplified model of computing jobs.*

failure rate).

Here, a simple model of computing jobs is used to show the effect of different parameters on performance. As shown in Figure 2.5, the model is as follows:

- Task Group "i" depends on the results of Task Group "$i-1$". Thus, all the Task Groups are processed in sequence.
- Each task has the same process time on different workers.
- The failure penalty is the time to detect a failure controlled by a time-out threshold.

ARR-Optimization Method

To find the proper value of *ARR*, two conditions of *ARR during the process of a task group* are discussed.

First Condition: During the process of a task group, ARR has to satisfy a condition that is limited by the number of workers. This condition can be described as follows:

$$ARR \times N_{incomplete} \leq W \tag{2.12}$$

$$\Rightarrow ARR \leq \left\lceil \frac{W}{N_{incomplete}} \right\rceil . \tag{2.13}$$

With W workers and $N_{incomplete}$ incomplete tasks, ARR cannot exceed the value.

Second Condition The purpose of redundant task dispatch is to insure that all the incomplete tasks have been dispatched ARR times, and therefore to increase the probability that all these tasks will be finished without failures. Assume that there are enough workers and all the incomplete tasks have been dispatch ARR times. The number of failures can be described as the following equation.

$$N_{incomplete} \times AFR^{ARR}. \tag{2.14}$$

By defining a threshold t for the number of failures, the second condition of ARR that can expect less than t failures is found as follows:

$$N_{incomplete} \times AFR^{ARR} \leq t \tag{2.15}$$

$$\Rightarrow ARR \geq \left\lceil \log_{AFR} \frac{t}{N_{incomplete}} \right\rceil . \tag{2.16}$$

In Equations (2.15) and (2.16), it is assumed that there are enough workers to process the duplicate copies of incomplete tasks. While this assumption is not satisfied, the *second condition* does not exist.

A Near Optimal ARR In this work, the "wasted time for failed task re-execution" and "wasted time for unnecessary redundant task dispatch" are considered as the two overheads. Thus, an optimal ARR should minimize the sum of these two overheads.

Because failures in a large scale dynamic volunteer computing platform cannot be predicted accurately, it is too difficult to give a function of this sum. Therefore, an approximate method is proposed to find a near optimal ARR. By setting the threshold t to a value less than 1, the "wasted time for failed task re-execution" can be considered as 0. The problem of finding the minimum value of the sum is simplified to be the problem to find the minimum value of "wasted time for unnecessary redundant task dispatch." Since the "wasted time for unnecessary redundant task dispatch" increases with ARR, a near optimal ARR should equal to the value found with the second condition in Equation (2.16). However, when there are no enough workers to process the duplicate copies of incomplete tasks, the second condition cannot be satisfied. Thus, the value found with the first condition is used.

The approximate optimization method is designed as follows:

1. If there are enough workers to process all the duplicate copies of incomplete tasks, ARR=$\left\lceil \log_{AFR} \frac{t}{N_{incomplete}} \right\rceil$.
2. Otherwise, ARR=$\left\lceil \frac{W}{N_{incomplete}} \right\rceil$.

To estimate AFR at the runtime, the dispatcher is enhanced to provide a recent AFR. This function logs the "Results" and "Task Failure" messages to a fixed length queue. Using this queue, the recent AFR is estimated as follows:

$$AFR = \frac{number\_of\_task\_failure}{number\_of\_task\_failure + number\_of\_results}. \tag{2.17}$$

2.4 Evaluation Results

The performance of the proposed mechanism is evaluated using a network simulator that is developed on a commercial network simulation tool - OPNET Modeler [35]. The purposes of this simulation are as follows:

1. To prove that the redundant task dispatch guarantees a high dependability.

2. To evaluate the mechanism with different parameters (AFR, ARR, Task Process Time, Failure Penalty, Number of Task Groups).

3. To demonstrate the effectiveness of the proposed runtime optimization method for the ARR.

2.4.1 The Simulator Configuration

The models of the dispatcher and the workers are shown in Figure 2.6. The simulation parameters are as follows:

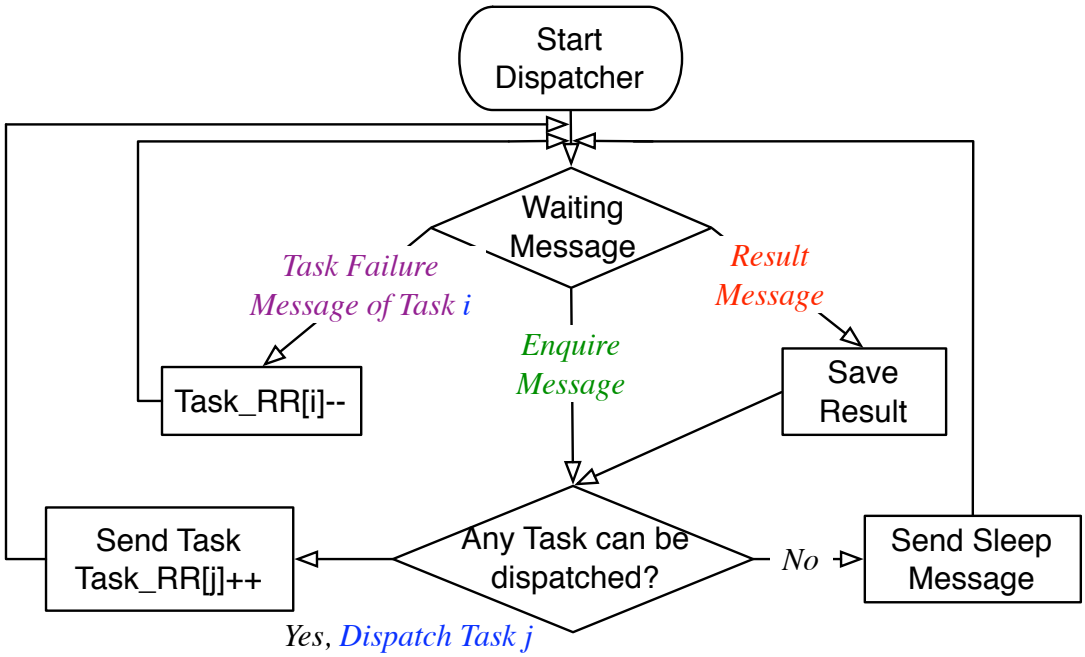
- Number of Workers: the number of workers in the network.
- Number of Tasks: the number of tasks for the computing job.
- Task Process Time: process time of a task.
- Failure Penalty: a time-out threshold to detect a task failure on workers.
- Idle Worker Inquire Interval: an inquire interval of idle workers.
- Number of Task Groups: the number of task groups in the computing job. It is the factor of inter-task dependency.
- AFR: an average failure rate of the tasks.
- ARR: an acceptable redundancy rate. The redundancy rate of any task cannot be larger than ARR.

The default parameters of the simulator are listed in Table 2.1.

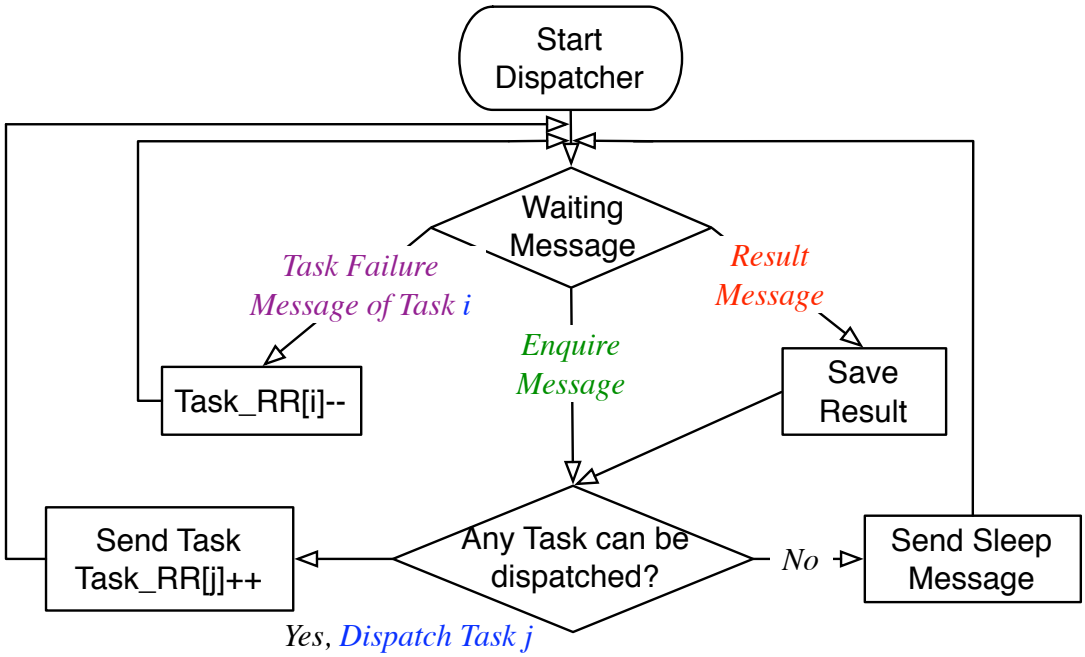
2.4.2 Performance Evaluation

Effectiveness of the Redundant Task Dispatch

Figure 2.7 shows the dependability achieved with and without redundant task dispatch, using the different number of task groups. The dependability values for redundant task



(a) Dispatcher model.



(b) Worker model.

Figure 2.6: *Simulation models of the dispatcher and the workers.*

Table 2.1: *Default parameters.*

Parameter	Value
Number of Workers	8,192
Number of Tasks	800,000
Task Process Time	120 seconds
Failure Penalty	150 seconds
Idle Worker Inquire Interval	60 seconds
Number of Task Groups	2, 4, 8, 16, 32, 64
AFR	0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6
ARR	1, 2, 4, 8, 16, 32, 64, 128

dispatch are calculated from the minimum total execution times among several simulation runs for the different ARR, the same AFR and *Number of Task Groups*. This dependability values can be considered as the optimal dependability values that can be achieved, using the redundant task dispatch.

For the different number of task groups with the same AFR of 60%, redundant task dispatch helps the mechanism to achieve a dependability of about 0.28 on the volunteer computing platform. The dependability without redundant task dispatch is always worse than the one with redundant task dispatch. Even with a less frequent failure rate, the results with redundant task dispatch are much better.

Without the redundant task dispatch, the dependability decreases as the number of task groups increases. The reason is that while increasing the number of task groups, there are more "blocked" status under frequent re-execution of tasks because of their failures. All the computing power of idle workers is wasted at the "blocked" status, while the computing power can be used to achieve a lower task failure rate. In an extreme case, with 64 task groups and 60% AFR, the dependability is only about 0.089, while the optimal dependability achieve 0.28 with the redundant task dispatch. From the above comparison, it is clear that the redundant task dispatch helps the volunteer computing platform to guarantee a high dependability.

Figure 2.7 also shows that for the same AFR, the optimal dependability hardly changes

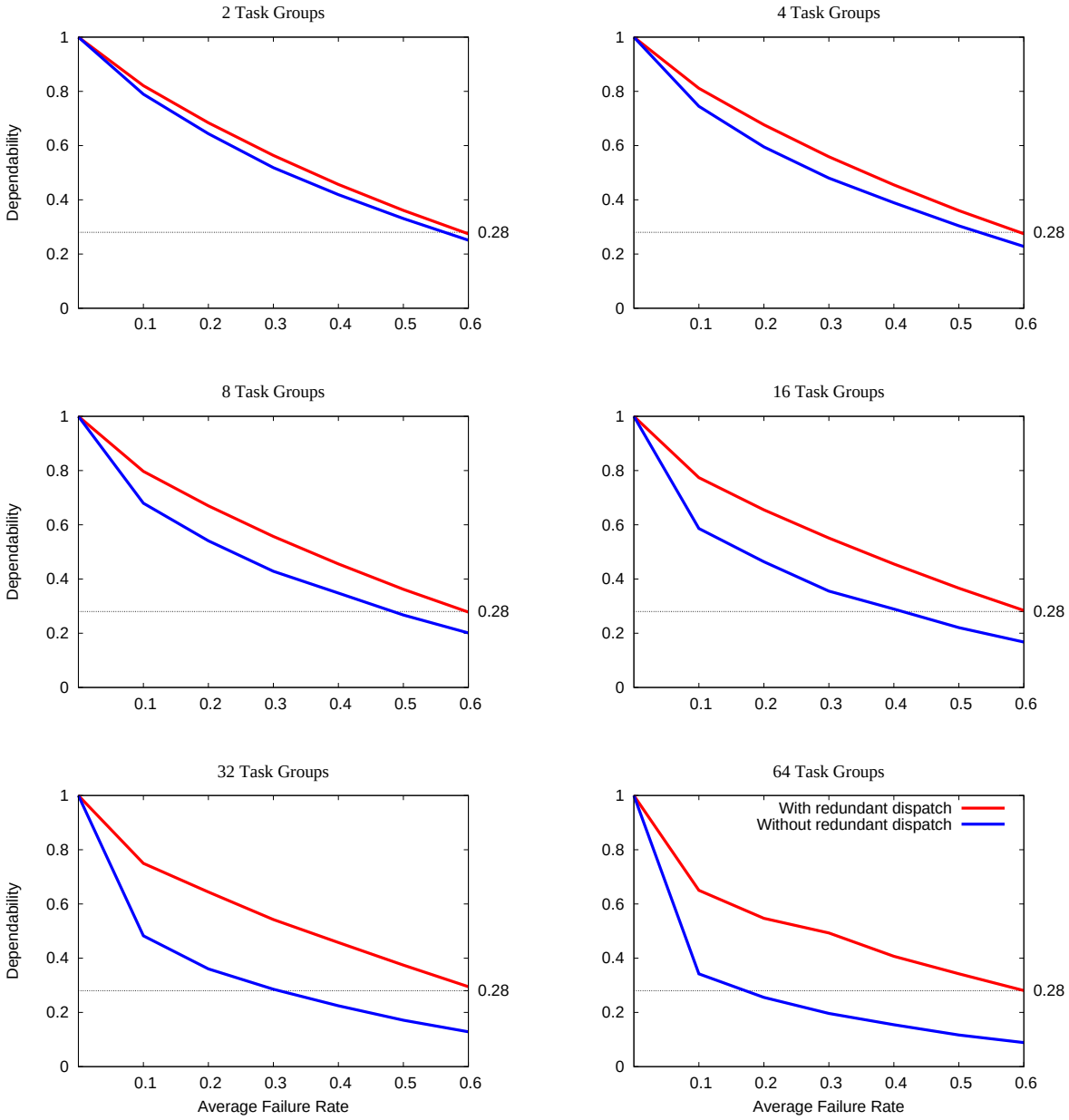


Figure 2.7: *The dependability achieved with and without redundant task dispatch.*

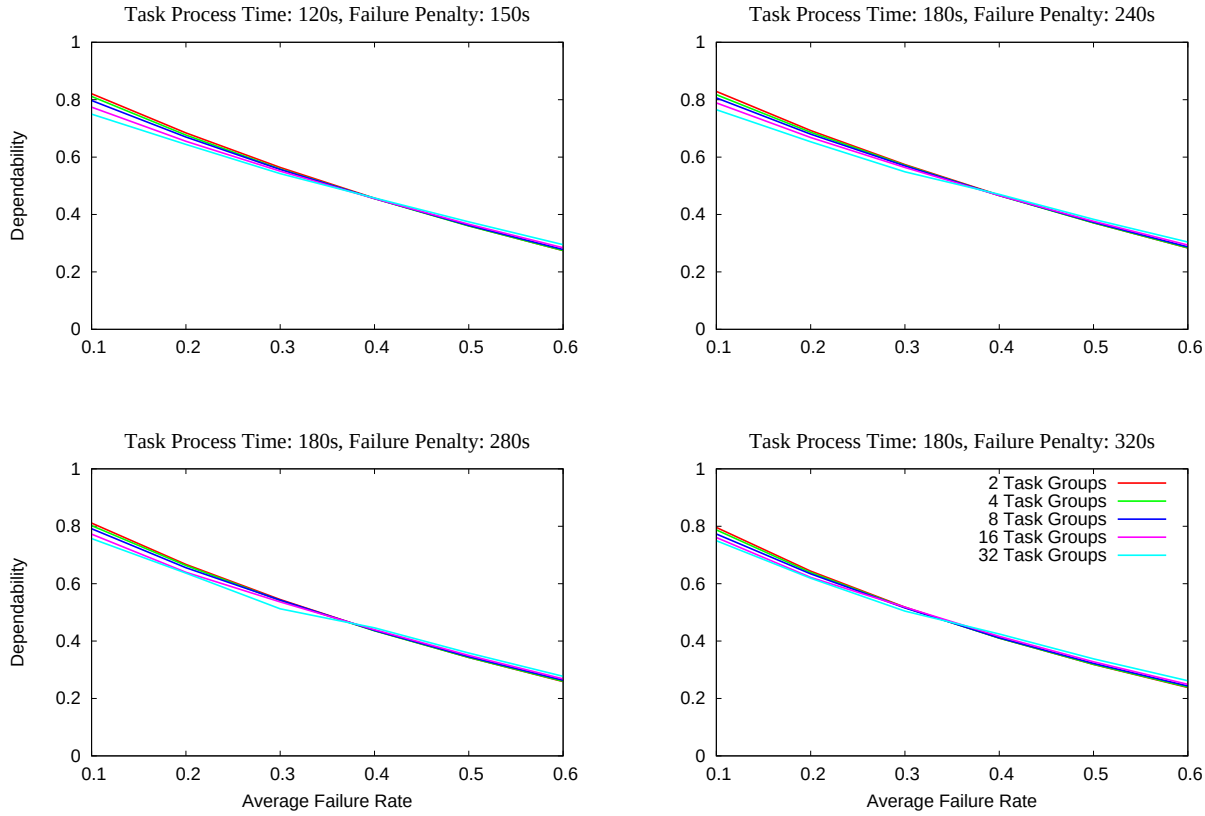


Figure 2.8: *Optimal dependability for different task process time and failure penalty.*

for the different number of task groups. To prove this, three more sets of simulation parameters (task process time and failure penalty) are evaluated by the simulator. Figure 2.8 compares the optimal dependability regarding the number of task groups with AFR for different sets of simulation parameters. The results indicate that the optimal dependability is independent of the number of task groups.

As demonstrated in Figure 2.8, the number of task groups hardly affects the optimal dependability. Therefore, it can be described as a function of two parameters, the ratio of *failure penalty* to the task process time and AFR as follows:

$$dependability_{optimal} = f_1(\frac{failure\_penalty}{task\_process\_time}, AFR). \tag{2.18}$$

Moreover, the dependability can be described as a function of the optimal dependabil-

ity and ARR.

$$dependability = f_2(dependability_{optimal}, ARR). \tag{2.19}$$

Equation (2.19) indicates that for a set of simulation parameters (task process time, failure penalty and AFR), the dependability depends only on ARR. Therefore, by estimating a near optimal value of ARR with the optimization method proposed in Section 2.3.5, the volunteer computing platform can achieve a near optimal performance.

Effect of ARR

Figure 2.9 shows that ARR has a great effect on the total execution time. While ARR is too small, with extremely volatile peers, the execution time is very long and thus the dependability is low. When ARR is too large, it also results in a low dependability, because too much computing power is wasted for redundant task process. In an extreme case, with 64 task groups and an AFR of 60%, the optimal dependability can be achieved with $ARR = 16$. The optimal dependability is 2 times better than the dependability with $ARR = 2$. For each simulation run, there is a proper value of ARR that helps the computing platform to achieve an optimal dependability. The optimal value depends on the simulation parameters. These results indicate that, in the case of a real world system, ARR has to be optimized at runtime estimating such parameters.

Simulation with Optimized ARR

The runtime optimization method is evaluated by the simulator. The parameters are listed in Table 2.2. In this simulation, AFR is defined only for the workers to simulate workers' failures. The dispatcher uses Equation (2.17) to estimate the runtime AFR.

Figure 2.10 shows the values found with the two conditions for the default simulation parameters during the processing of a task group. The simulation results are shown in Figures 2.11 and 2.12. The lower x-axis represents AFR. The upper x-axis represents the number of task groups. For the same AFR, the results are listed from left to right in

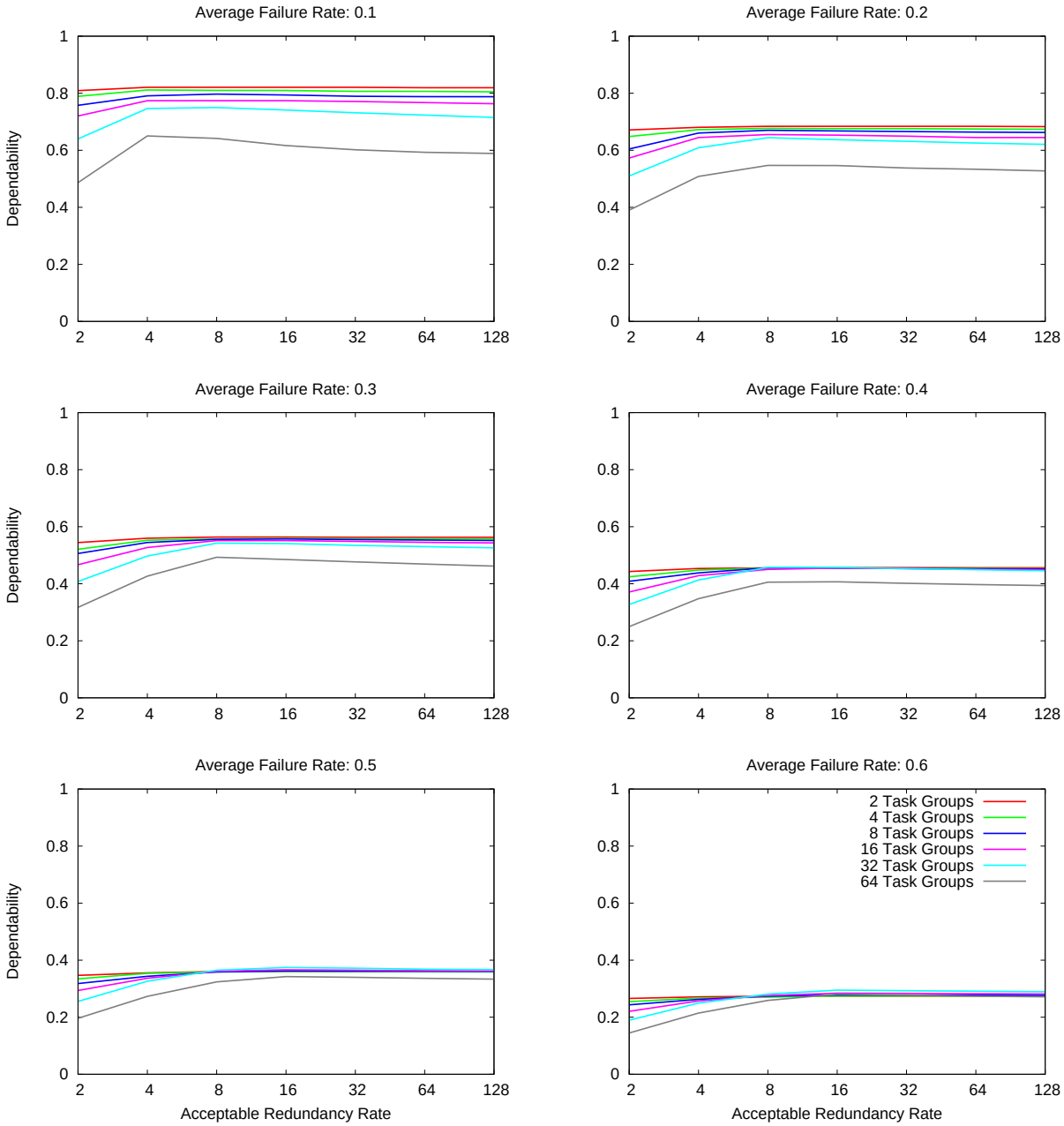


Figure 2.9: *Effect of ARR for different simulation parameters.*

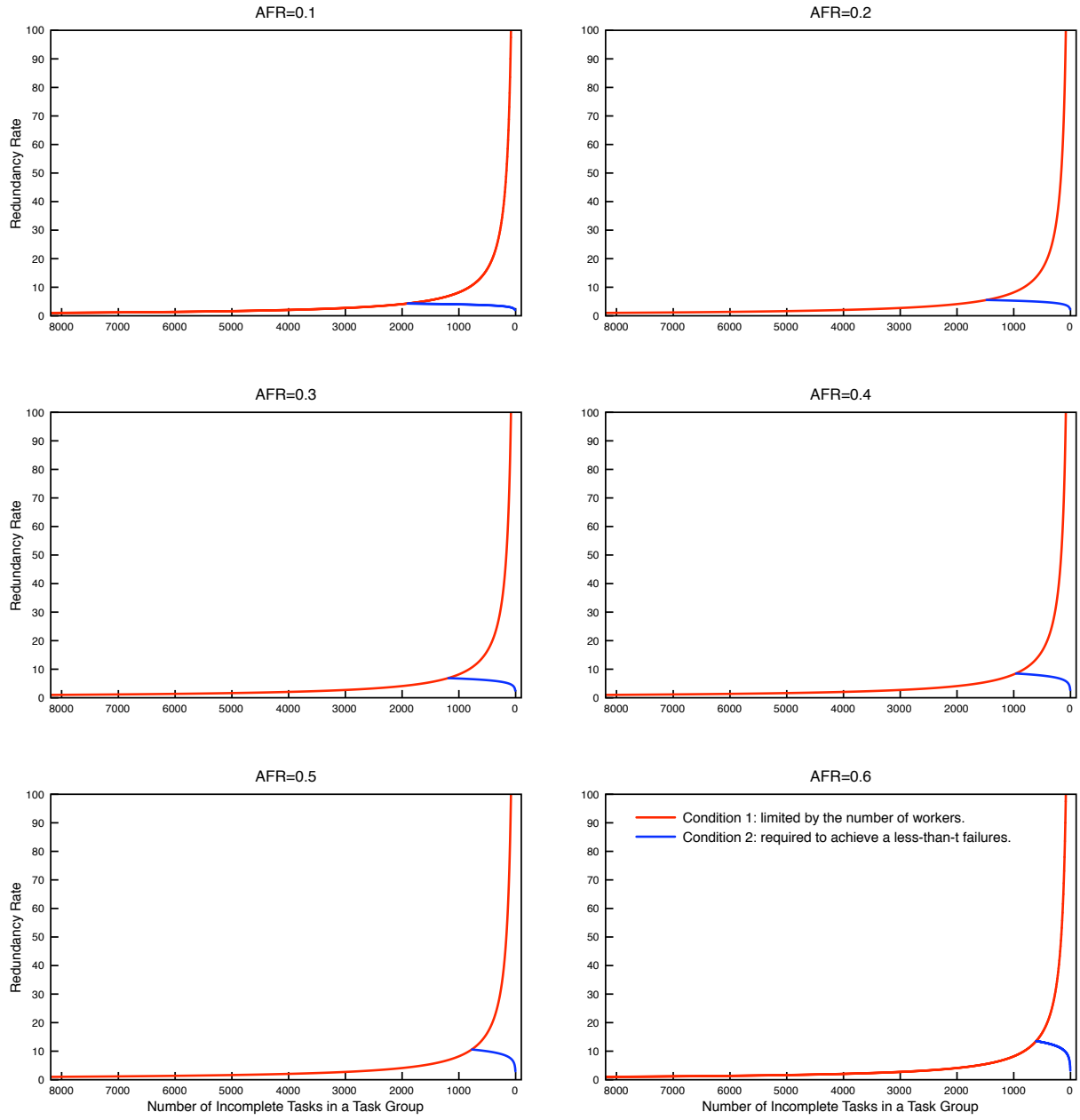


Figure 2.10: *The two conditions of ARR with 8192 workers in one task group.*

Table 2.2: *Simulation parameters for runtime optimization method.*

Parameters	Value
Task proces time, failure penalty	120s, 150s 180s, 240s 180s, 280s 180s, 320s
AFR	0.1, 0.2, 0.3, 0.4, 0.5, 0.6
Number of task groups	2, 4, 8, 16, 32, 64
Threshold t	0.1
Message queue size	10000

the order of the number of task groups. The total execution time and the number of task dispatches are normalized by the corresponding results of the simulation with optimal dependability. All the results with optimization are very close to ones with optimal dependability. These results prove that the ARR found by the runtime optimization method is a near optimal value.

Performance Evaluation Under Heterogeneous Environments

In the previous simulations, the homogeneous computing workers are assumed, and tasks processed on different workers have the same execution time that is predefined as a parameter of *task process time*. A large scale volunteer computing platform in the real world is a heterogeneous environment. In a more realistic environment, since different workers have different performance, and therefore the task process time on each worker is different. To examine the performance in a heterogeneous environment, the average value of task process times of all workers is defined as the average task process time ($ATPT$). To simulate a heterogeneous environment, the task process time on each worker is randomly drawn from a uniform distribution ranging from (0.5x$ATPT$) to (1.5x$ATPT$).

The simulation results of the heterogeneous environment are shown in Figures 2.13 and 2.14. The y-axis is the normalized total execution time that is normalized by the results under the homogeneous environment, in which the performance of each worker is

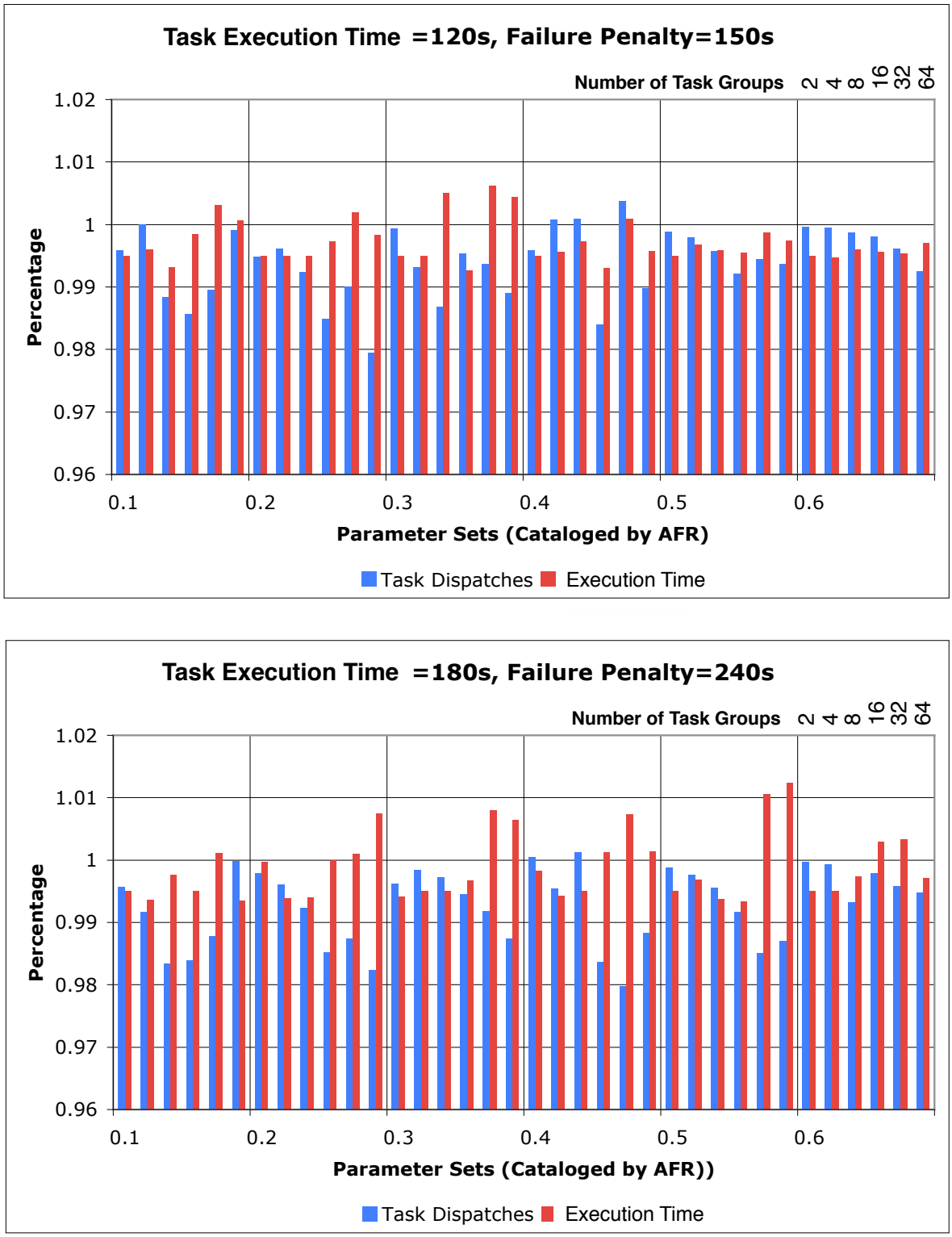
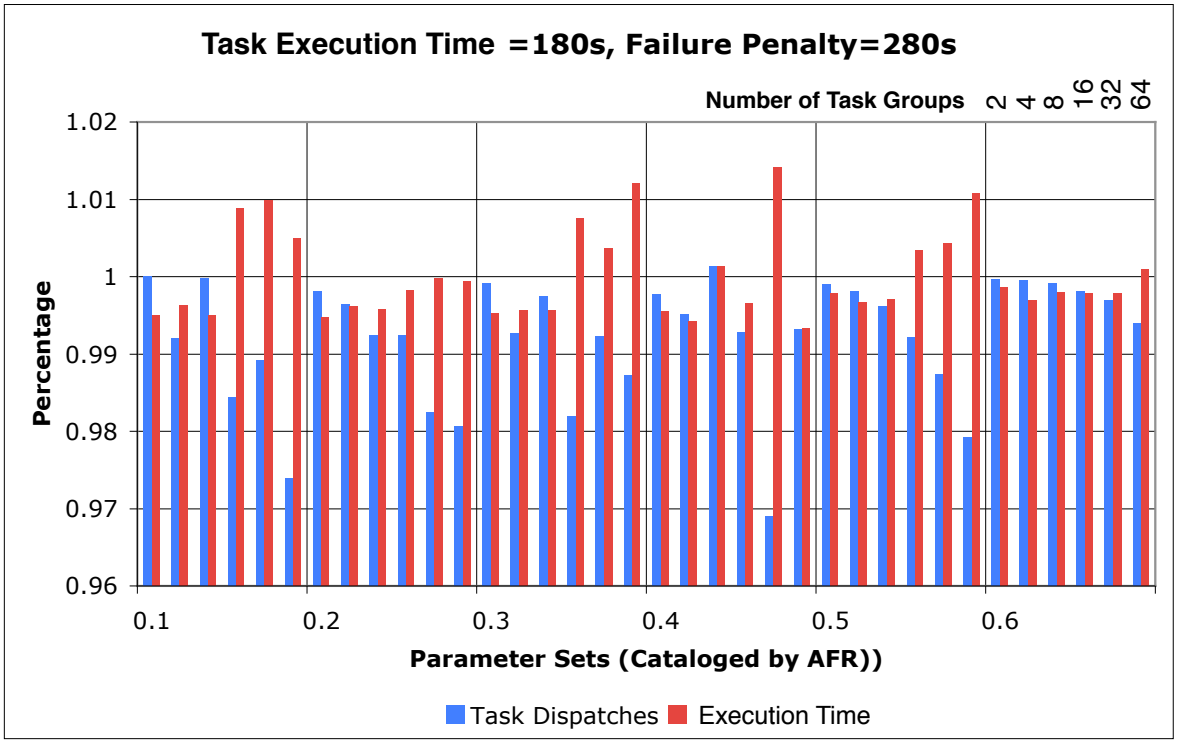


Figure 2.11: *Simulation results with optimization method (1).*



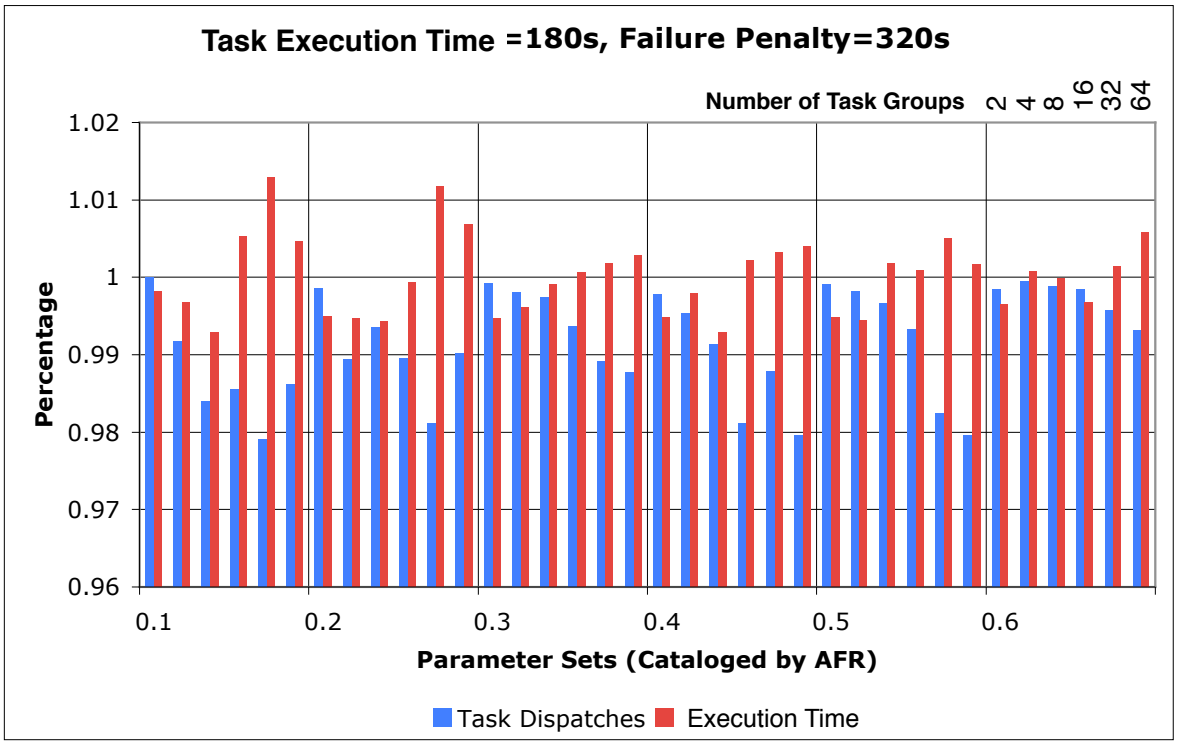


Figure 2.12: *Simulation results with optimization method (2).*

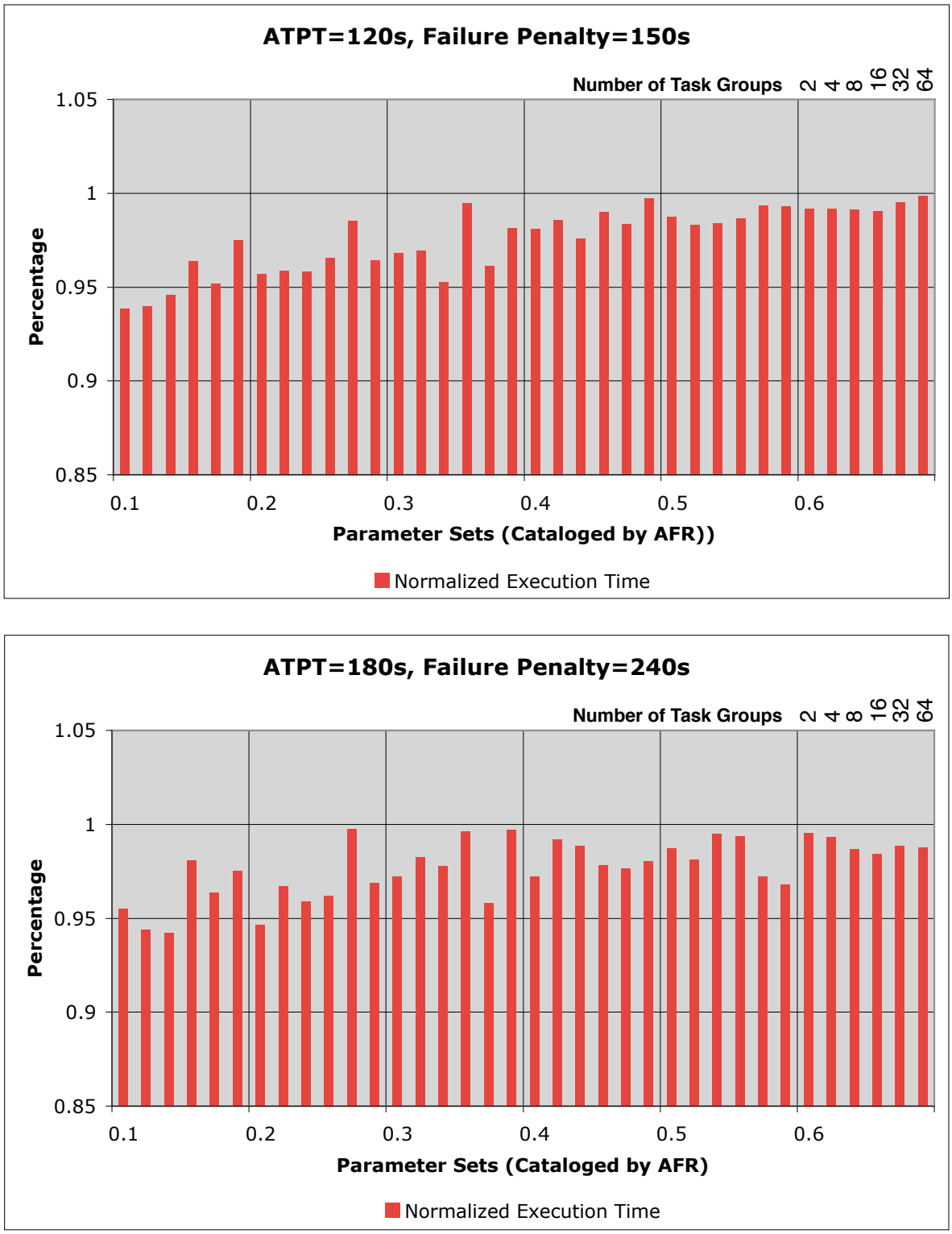


Figure 2.13: *Simulation results under heterogeneous environments (1).*

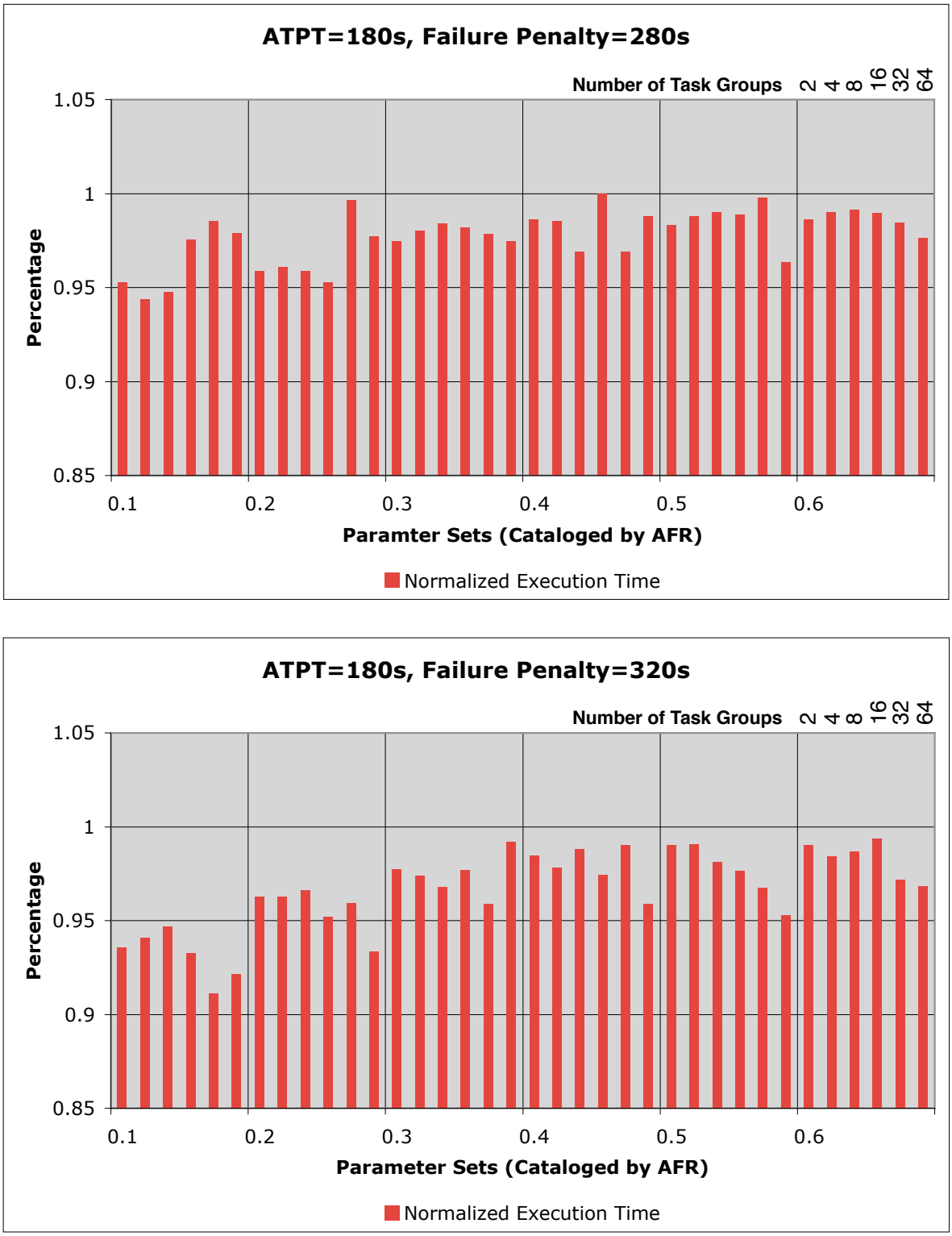


Figure 2.14: *Simulation results under heterogeneous environments (2).*

the same as $ATPT$. The figure uses the same x-axis configuration as Figures 2.11 and 2.12. The figures show that the results of the heterogeneous environment are faster than those of the homogeneous environment even the average task process times are the same in these two environments. The reason is that the status of a task is set to "complete" when the first result from one of workers is received in the heterogeneous environment. In the heterogeneous environment, half of the workers processing the same task are expected to return a result earlier than $ATPT$. Therefore, the probability that a task in the heterogeneous environment can get a result earlier than $ATPT$ is equal to 50%.

Suppose that there are x workers processing one task. Then, there are $\frac{x}{2}$ workers that are expected to return a result earlier than $ATPT$. The probability that the status of this task is set to "complete" earlier than $ATPT$ is $1 - AFR^{\frac{x}{2}}$. Therefore, even when $AFR = 0.6$, the probability will be higher than 50% with at least four workers. As a result, the total execution time becomes shorter. These results prove that the dependable workflow management mechanism can efficiently work even in a practical large scale volunteer computing platform.

2.5 Conclusions

Volunteer computing is a promising technology that has the potential to provide more computing power than any supercomputer, cluster or grid. However, the applicable areas of existing volunteer computing platforms are limited to embarrassingly parallel problems. None of the relate work in the field of grid workflow management is suitable for a volunteer computing platform with volatile peers. Therefore, a dependable workflow management mechanism has been proposed for volunteer computing platforms. The two issues that differentiate this work from the existing volunteer computing platforms are as follows:

- Workflow management mechanism: extends the applicable areas.
- Redundant task dispatch and runtime optimization: guarantees high dependability

(i.e. low performance degradation) even with highly volatile peers.

The proposed dependable workflow management mechanism has been evaluated using network simulation. The large scale simulation results indicate that the redundant task dispatch guarantees a high dependability even with extremely volatile peers. With the optimization method proposed in Section 2.3.5, the computing platform can achieve a near optimal dependability. In addition, a heterogeneous volunteer computing platform has been evaluated in terms of the total execution time. The simulation results indicate that the dependable workflow management mechanism works efficiently, even in a heterogeneous environment.

Chapter 3

Performance-Oriented Task Dispatch Policy

3.1 Introduction

In this chapter, a performance-oriented task dispatch policy is proposed. Although the redundant task dispatch policy proposed in Chapter 2 has shown a significant performance improvement compared to the non-redundant one, it has a major limitation: the average failure rate model is not the best fit for the volunteer peers in the real world. Thus, this chapter extends the policy so as to address the limitation.

A performance-oriented task dispatch policy based on the failure probability estimation is proposed in this chapter. A heuristics-based mechanism for failure probability estimation is proposed based on a life cycle model of volunteer peers and the statistical data. The tasks with the highest failure probabilities are selected for dispatch when multiple task enquiries come to the dispatcher. The estimated failure probability is used to find the optimized task assignment that minimizes the overall failure probability of these tasks. This performance-oriented task dispatch policy is evaluated with two real world trace data sets on a simulator. Evaluation results demonstrate the effectiveness of this policy. The effects of parameters are also discussed.

3.2 Related Work

The failure probability is estimated based on the analysis of peer availability data. The resource availability problem has been studied a lot for cluster, servers, PCs in a corporate network, grid, and volunteer computing systems.

3.2.1 Statistical Resource Availability Characterizing

There have been a large number of works on the problem of characterizing resource availability statistically.

Root Cause Analysis of Failures

Root cause analysis of failures has been studied in [36–39]. The software-related failure is reported to be around 20% [38], 50% [36, 37], and from 5% to 24% [39]. The percentage of hareware-related failure is from 10% to 30% in [36–38], and from 30% to over 60% [39]. The network-related failure is significant in some of the works, while it accounts for around 20% [37] and 40% [38] of the failures. Human errors also lead to 10% - 15% [36] and 14% - 30% [38] of the failures. These works reported different breakdown of failures, because of the different systems they studied.

Fitting Distribution to Empirical Availability Data

Some other works study the statistical distributions of empirical availability data, e.g. *Time-to-Fail (TTF)* and *Down Time (DT)*. Such methods find the best fitted theoretical distribution for a given empirical data, by estimating the parameters of the theoretical distributions with techniques such as Maximum Likelihood Estimation (MLE) [40]. Several distributions have been used to model the peer availability, including lognormal, Weibull, exponential, hyper-exponential, and Pareto distributions. The detail of these distributions and their properties can be found in [41].

Exponential distribution and hyper-exponential distribution have been used to study the availability behaviors of software, operating system, workstation, and peer-to-peer file sharing system in [42–47]. For the research such as process lifetime estimation [48] and network performance [49], Pareto distribution has been used a lot. Weibull distribution is another widely used distribution for modeling the resource availability. Xu et al. [50] applied it for the modeling of network connected PCs.

Several studies [39, 51–53] compared different distributions for the modeling. Nurmi et al. [51, 54] used exponential, hyper-exponential, Weibull, and Pareto distributions to model the *TTF* availability data gathered from student lab computers, a cycle-harvesting distributed computing system - Condor [55, 56], and an early survey of Internet hosts [57]. Goodness-of-fit analysis indicated that hyper-exponential and Weibull distributions fit the empirical data more accurately. Schroeder et al. [39] studied the distribution fitting of *TTF* in high-performance computing (HPC) systems with 4750 machines, using Welbull, lognormal, gamma, and exponential distributions. The results pointed out that Weibull distribution is a better fit. Iosup et al. [52] found Weibull the best fitted among several distributions for *Mean Time Between Failure (MTBF)* and failure duration data of Grid'5000 [58, 59].

More recently, Nadeem et al. [53] also applied several distributions to the analysis of grid resource availabilities. It introduced the class level modeling method by identifying three types of resources in the Austrian Grid [60]. Based on the administration policy, it categorized the resources into three classes: dedicated resources, temporal resources and on-demand resources. The distribution fitting and goodness-of-fit test are done separately for each class's availability (*TTF*) and unavailability (*Mean Time to Reboot (MTR)*) data. While other works found one or two best fitted distributions, this work found different best fitted distributions for different class.

3.2.2 Availability Prediction

Nurmi et al. [54] assumed a homogeneous environment, and proposed an availability prediction method on top of the found Weibull distribution. This method answered the question what is the largest availability duration for a given confidence value and a desired percentile. Iosup et al. proposed a resource availability model [52] that considered the failure distribution among clusters, the *TTF* distribution, failure duration distribution, and failure size (number of processors) distribution. This model is used to predict the failures in a multi-cluster grid system.

Some other works [61, 62] utilized the availability pattern on weekdays and weekends to predict the availability. Nadeem et al. [53] used Bayes' Rule and Nearest Neighbor Rule to predict the resource availability. Mickens et al. [63] proposed saturating counter predictors, state-based history predictors, a linear predictor, and a hybrid predictor that dynamically selects the best predictor. These predictors have been evaluated with trace data sets of distributed servers, peer-to-peer network, and corporation PCs.

3.3 A Heuristics-based Failure Probability Estimation

These prediction methods of resource available status reviewed in the last section provide a different accuracy for their selected environments. Because this chapter targets at finding optimized task assignment with estimated task failure probabilities, the distribution of empirical availability data can provide enough information. Here, a simple and straight heuristics-based failure probability estimation method is employed.

3.3.1 Life Cycle of a Volunteer Peer

The life cycle of a volunteer peer can be modeled as shown in Figure 3.1. *TTF* is the time between a peer's start/restart and the next failure/shutdown. *DT* is the time between a

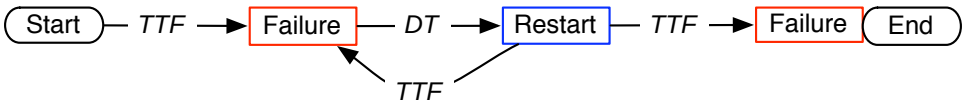


Figure 3.1: *Life cycle of a volunteer peer.*

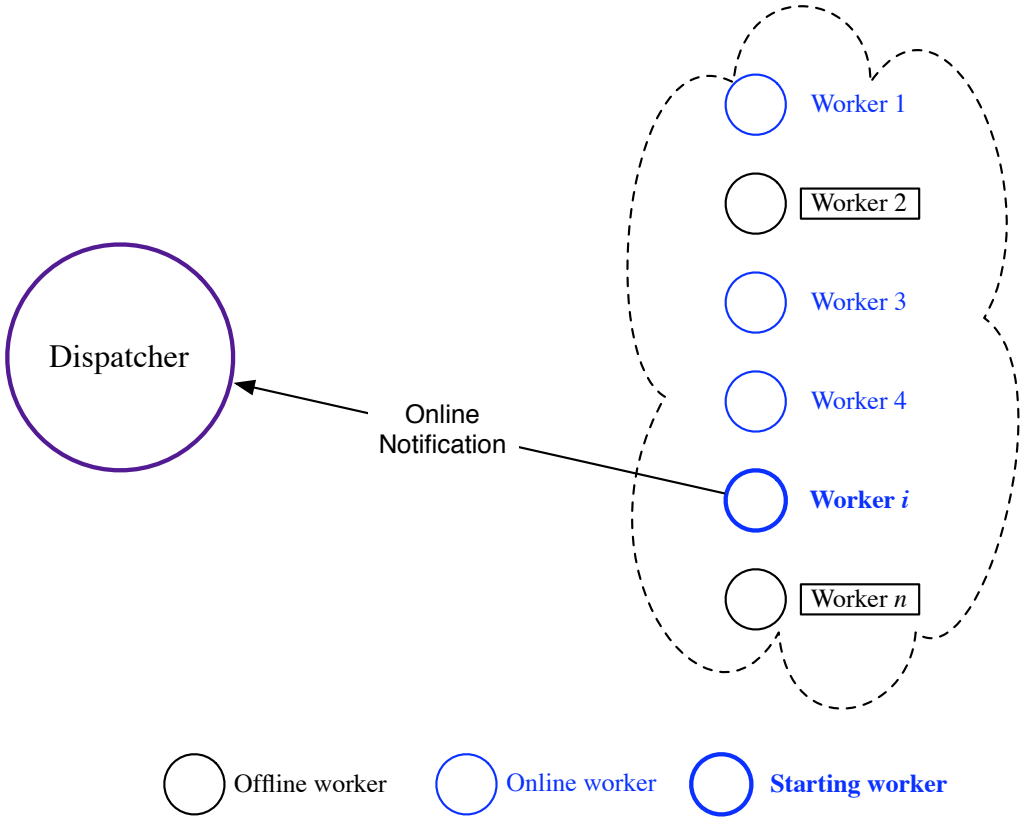
failure and the next peer restart. Given a statistical distribution of *TTF*, the *cumulative distribution function (CDF)* of this distribution's value at each uptime x is the probability that a peer's *TTF* is smaller than or equal to x, which equals to the failure probability at uptime x. The failure probability monotonously increases with the time. Since none of a single distribution can characterize the resource availability accurately for any systems in large scale computing environments [51, 53], a heuristics-based mechanism is proposed to estimate the failure probability at runtime with the gathered *TTF* data.

3.3.2 Failure Probability Estimation

Volunteer computing platforms has two kinds of peers: dispatchers and workers. A task dispatcher is a specific server that controls a volunteer computing platform. Workers are volatile peers that compute tasks and send back the task results to the dispatcher. To estimate the failure probability, the runtime *TTF* data are required. To gather such runtime data, a worker availability status list is maintained by the dispatcher. The list stores the start time of each worker. If a worker is currently unavailable, it is marked as *offline* in the list. The list is maintained as follows:

A Worker Goes Online As shown in Figure 3.2(a), when a worker goes online, it sends an online notification message to the dispatcher. Once the notification is received, the dispatcher updates the worker availability status list as shown in Figure 3.2(b). The current time is stored as the start time of this worker.

Find Offline Worker To gather the runtime *TTF* data, the dispatcher also checks the availability status of workers periodically. As shown in Figure 3.3(a), the dispatcher sends



(a) Worker i goes online, sends an online notification to the dispatcher.

Worker Availability Status List

Worker ID	Start time
1	2008/11/22 7:20
2	offline
3	2008/11/21 5:50
4	2008/11/23 3:10
...	...
i	offline
...	...
n	offline

Received online notification

Worker ID	Start time
1	2008/11/22 7:20
2	offline
3	2008/11/21 5:50
4	2008/11/23 3:10
...	...
i	2008/11/23 13:00
...	...
n	offline

Current time

(b) The status of worker i in the worker availability status list changed, after dispatcher received the notification.

Figure 3.2: *Worker i goes online.*

status checking messages to the workers that are marked online in the worker availability status list. Once the message is received by an alive worker, the worker sends a reply message back to the dispatcher as shown in Figure 3.3(b). If a worker is offline, it cannot reply the checking message. Then, it is marked as *offline* in the list. As an example, before the periodical status check, worker 4 in Figure 3.3 has been marked as *online* with a start time in the list, and then went offline. Thus, it does not reply the checking message. The dispatcher then updates the worker availability status list, and marks worker 4 to be *offline*. It also calculates the TTF of the worker 4's last online session. Given the current time and start time of the last online session, the TTF is 680 minutes (from 2008/11/23 3:10 to 2008/11/23 14:30).

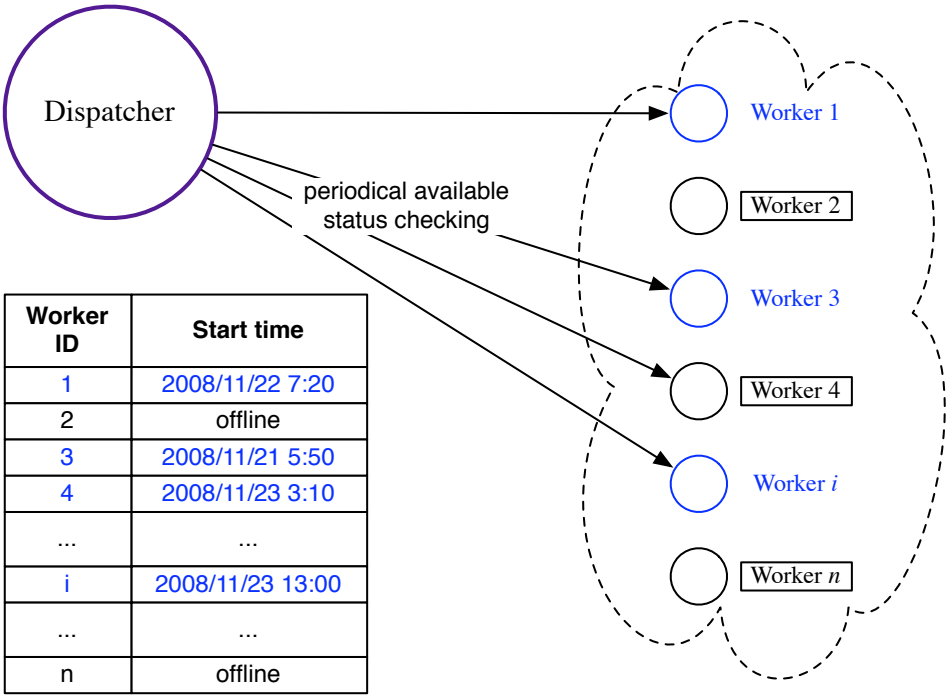
With this simple periodical availability status checking mechanism, the runtime TTF data are gathered on the dispatcher. Thus, the TTF distribution can be found at the runtime. Suppose the gathered TTFs are $\{ttf_1, ttf_2, ttf_3...ttf_n\}$, where n is the number of gathered TTFs. The failure probability $F(x)$ of a worker (x is the time after a worker went online) can be estimated as shown in Equation (3.1):

$$F(x) = \frac{n_x}{n}, \tag{3.1}$$

where n_x is the number of TTFs that less than or equal to x.

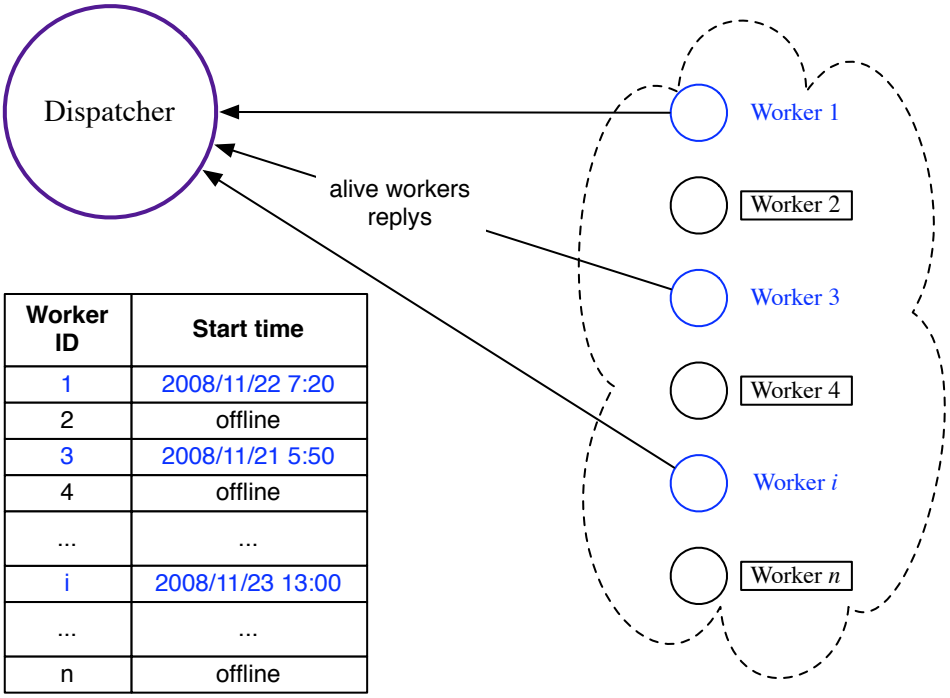
3.4 Least Failure Probability Dispatch Policy

With the failure probability estimation, a performance-oriented task dispatch policy - *Least Failure Probability Dispatch (LFPD)* for volunteer computing platforms is proposed. The assumptions are slightly different from the ones in Chapter 2. While Chapter 2 assumes a homogeneous environment, this chapter assumes that the volunteer computing platform is a heterogeneous environment that all the workers have different performance and different bandwidth to the dispatcher.



Worker ID	Start time
1	2008/11/22 7:20
2	offline
3	2008/11/21 5:50
4	2008/11/23 3:10
...	...
i	2008/11/23 13:00
...	...
n	offline

(a) Dispatcher periodically checks the availability status of workers that are marked online in the worker availability status list.



Worker ID	Start time
1	2008/11/22 7:20
2	offline
3	2008/11/21 5:50
4	offline
...	...
i	2008/11/23 13:00
...	...
n	offline

Worker 4 *TTF* = 2008/11/23 14:30 - 2008/11/23 3:10 = 680 minutes

Current time

(b) Worker 4 is found to be offline, because it fails to reply the check message.

Figure 3.3: *Checking worker availability status, gathering* TTF *data.*

3.4.1 An Enhanced Workflow Management Mechanism

A workflow management mechanism has been proposed in Chapter 2. It is responsible for directing the workflow control and the task information update. It cannot fully satisfy the requirement of the *LFPD*, because it assumed a same task failure probability for all the dispatched tasks. Thus, an enhanced workflow management mechanism is proposed to assist the *LFPD*.

To support the *LFPD*, the following information of each task i is stored and updated by the dispatcher.

- Status: "undispatched", "dispatched", and "finished."
- Redundancy rate that records how many workers process the task i at the same time: RR_i.
- The list of worker IDs that process the task i: $workerID_i[RR_i]$.
- The list of estimated failure probability for each copy of task i: $EFPs_i[RR_i]$.
- The overall failure probability: FP_i.

The overall failure probability is calculated as:

$$FP_i = \prod_{k=1}^{RR_i} EFPs_i[k]. \tag{3.2}$$

Similar to the original workflow management mechanism, the enhanced workflow management mechanism uses the status information to analyze whether an "undispatched" task can be dispatched or not. An "undispatched" task can only be dispatched when all the tasks that it depends on are "finished." A workflow has two kinds of status: "blocked" and "unblocked." While there is no such "undispatched" task, the workflow management mechanism uses the redundant task dispatch to reduce the performance degradation. Such status of a workflow is defined as "blocked."

The initial workflow information of each task i is as follows:

- Status: "undispatched."
- Redundancy rate: $RR_i = 0$.
- The overall failure probability: $FP_i = 1$ which means that a task will never finish before it is dispatched.

When the dispatcher dispatches a task i to a worker j, it provides the required input values from the preceding tasks, and then changes the task i's status to "dispatched" if it was "undispatched" and increases RR_i by one. The worker j is stored in the $workerID_i[RR_i]$. The failure probability of this assignment $\{task\ i \rightarrow worker\ j\}$ is estimated and stored in $EFPs_i[RR_i]$. The overall failure probability is calculated again.

When the dispatcher receives a result of task i from a worker j, it changes the status of task i to "finished", and sends a "cancel" message to the workers in $workerID_i[RR_i]$, except worker j. The function to cancel duplicate copies after the task finish can reduce the overhead for redundant task dispatch.

When a worker j is found to be offline by the periodical available status check, RR_i of this task is decreased by one. The worker ID is removed from $workerID_i[RR_i]$. Finally, the overall failure probability of the task is updated.

3.4.2 The Task Selection and Dispatch Policies

While the workflow management mechanism controls the process of a job workflow, it requires policies to select the tasks for dispatch, and find the task-to-worker assignment when the task enquiries come.

Task Selection Policy

In Chapter 2, *least-RR-selected* policy has been proposed to equally reduce the failure rate of all the "dispatched" tasks. It selects a task with the least redundancy rate and dispatches this task to the idle worker. Because *least-RR-selected* assumed a constant

task failure rate, it cannot be applied directly to the *LFPD*. In this chapter, therefore, a *highest-failure-probability-selected* policy is proposed to provide the similar function for *LFPD*. It selects the task with the highest overall failure probability.

Furthermore, the failure probabilities of a task on different workers are different in a heterogeneous environment. By considering the task assignment of multiple tasks to multiple workers, a lower overall failure probability can be achieved. Thus, the idea of *dispatch window* is introduced. The dispatch window is the number of tasks that will be dispatched together. Given a window size w, the dispatcher waits for task enquiries from workers until the dispatch window is full, then it selects w tasks with the *highest-failure-probability-selected* policy.

Dispatch Policy

After getting the w tasks to dispatch, the dispatcher finds the optimal task-to-worker assignment that minimizes the overall failure probability of the w tasks.

Suppose that each task i has its computation cost (cmp_i) and communication cost ($comm_i$) information, and each worker j has its performance ($Perf_j$) and bandwidth ($Band_j$) information. This information is available for the dispatcher. Given a task i and worker j, the estimated process time of task i on worker j is:

$$T_{i,j}^{EPT} = \frac{cmp_i}{Perf_j} + \frac{comm_i}{Band_j}. \tag{3.3}$$

Thus, the estimated failure probability of this assignment is:

$$EFP_{\{task\ i \rightarrow worker\ j\}} = F(T_{i,j}^{EPT} + Current\ Time - Start\ Time_j), \tag{3.4}$$

where $F(x)$ is the *CDF* of *TTF*'s distribution; $Start\ Time_j$ is the start time of worker j in the worker availability status list.

Suppose that the window size is w, the selected tasks are $\{t_1, t_2, t_3 \ldots t_w\}$, and the

worker peers in the dispatch windows are $\{p_1, p_2, p_3 \ ... \ p_w\}$. For each permutation of $\{p_1, p_2, p_3 \ ... \ p_w\}$, there is an assignment. For example, the following assignment is for the permutation $\{p_1{}', p_2{}', p_3{}' \ ... \ p_w{}'\}$:

$$\left\{ \begin{array}{c} t_1 \rightarrow p_1{}' \\ t_2 \rightarrow p_2{}' \\ t_3 \rightarrow p_3{}' \\ ... \\ t_w \rightarrow p_w{}' \end{array} \right\}. \tag{3.5}$$

For each of the assignment, the estimated failure probability *(EFP)* of each task-to-worker pair is calculated with Equation (3.4). Then, the overall failure probability of the assignment is:

$$OFP = \prod_{k=1}^{w} EFP_{\{t_k \rightarrow p_k{}'\}}. \tag{3.6}$$

There are $w!$ possible assignments. The dispatcher calculates each assignment's OFP, and compares them. The assignment with the least OFP is used for the task dispatch. The dispatcher sends tasks to the workers in the dispatch window, using the decided assignment. The workflow information is updated using the enhanced workflow management mechanism proposed in Section 3.4.1.

3.5 Evaluation Results

The effectiveness of the proposed *LFPD* is evaluated using a simulator that is developed on a discrete event simulation environment - OMNeT++ [64]. The purposes of this simulation are as follows:

1. To prove the effectiveness of the *LFPD* policy.
2. To verify the effect of the task dispatch window.
3. To study the effect of different parameters.

4. To analyze the effect of identifying multiple worker types.

3.5.1 Baseline Policies

To discuss the effectiveness of the *LFPD* policy, two baselines are used.

Simple Redundant Task Dispatch Policy

The window-size-1 is a special case of the *LFPD* policy, because there is only one (1!) task-to-work assignment. Thus, the window-size-1 *LFPD* policy can be considered as an extension of the original redundant task dispatch policy that uses the proposed heuristics-based failure probability estimation model. This simple redundant task dispatch policy is used as a baseline to discuss the effectiveness of the *LFPD* policy.

Greedy Dispatch Policy

The proposed *LFPD* policy selects task-to-worker assignments with least overall failure probability. Thus, the effectiveness of the *LFPD* policy highly depends on the estimation accuracy of the failure probabilities. If the dispatcher can predict task failures perfectly, it can eliminate all the task failures. In such case, an intensively optimized dispatch policy for volunteer computing platforms can be achieved. The comparison between such a dispatch policy and the *LFPD* policy can demonstrate the effectiveness of the *LFPD* policy. Therefore, in this chapter, a greedy dispatch policy that can predict failure perfectly is used as another baseline in the following evaluation.

The greedy dispatch policy assumes that the dispatcher knows the perfect knowledge of the workers' future availability status. Using such knowledge, the dispatcher can perfectly predict whether a task can finish on a worker without failure. The way to find the best task-to-worker assignment is similar to the *LFPD* policy. Instead of using the assignment with the least overall failure probability, the greedy dispatch policy uses the assignment with the least number of failures.

Using the *LFPD* policy, the computing power is wasted in some cases. These cases can be found in advance with the knowledge of the worker's future availability status. The greedy dispatch policy adopts new rules to handle such cases as follows:

1. A task copy will incur a failure on a worker. The computing power of this worker is wasted. The greedy dispatch policy do not dispatch such tasks that will incur failures. A "sleep" message is sent to the worker. The worker sleeps for a pre-defined period after receiving the message.

2. A task copy will finish on a worker. Multiple copies of this task are running on different workers. However, this copy will finish later than some other copies (larger estimated finish time). A task copy's estimated finish time can be calculated when it is dispatched as: $EFT = T^{EPT} + Current\ Time$. The computing power of this worker is also wasted, because it does not contribute to the process of the job. Therefore, instead of dispatching duplicate task copies, the greedy dispatch policy insures that there is only one copy of any task. This old copy of a task is continually replaced with a new copy that has a smaller EFT value, whenever a new task-to-worker assignment is found for the workers in the dispatch window. When an old task copy is replaced with a new one, the old copy is canceled on the worker that executes it. If the new task copy has a larger EFT value, a "sleep" message is sent to the worker.

3.5.2 The Simulator Configuration

The dispatcher and worker modules are implemented with the OMNeT++ [64] to simulate both the proposed *LFPD* policy, the simple redundant task dispatch policy (the window-size-1 *LFPD* policy), and the greedy dispatch policy. To study the effectiveness of the policies in a real world environment, two real world resource availability trace data sets are used to generate the worker failures. The Skype trace data set [65] has the application-level resource availability data of 2,081 Skype supernodes for about 28 days. Skype is a

peer-to-peer VoIP software that connects thousands of volatile peers. The Microsoft PCs trace data set [66] stores the availability data of 51,662 desktop PCs within Microsoft Corporation network for 35 days. The volatile peers in the peer-to-peer network and desktop PCs in the corporation network are two typical worker types for the volunteer computing.

In a heterogeneous environment, the performance of each worker is different. To simulate such an environment, worker's performance parameters are generated with a power-law distribution. Because this work focuses on the computation-intensive problems that satisfy "*computation time* $\gg$ *data transfer time*," the communication cost is not considered in the simulation.

The simulation parameters are as follows:

- Number of Workers: the number of workers in the platform. It is the number of peers in the trace data sets.
- Number of Tasks: the number of tasks for the computing job.
- Mean Task Process Time: mean process time of a task. The process time of a task on a worker depends on the performance parameter of his worker.
- Idle Worker Inquire Interval: an inquire interval of idle workers that received the "sleep" message. The "sleep" message is only used in the greedy dispatch policy.
- Number of Task Groups: the number of task groups in the computing job. It is the factor of inter-task dependency.

The parameters of the simulator are listed in Table 3.1.

Because the same mean task process times are used to evaluate the two trace data sets, the different availability characteristics make it hard to compare the evaluation results of these two trace data sets. Thus, the Microsoft PCs trace data set is modified to have the same mean *TTF* as the Skype trace data set. The basic statistical properties of these two trace data sets are shown in Table 3.2.

Table 3.1: *Simulation parameters for* LFPD *policy and the greedy dispatch policy.*

	Skype Trace	Microsoft PCs Trace
Number of Workers	2,081	51,663
Number of Tasks	80,000	2,000,000
Mean Task Process Time	1250, 2500, 5000, 10000 (s)	
Number of Task Groups	5, 10, 20	
Idle Worker Inquire Interval	200 seconds	

Table 3.2: *Summary of the basic statistical properties of the Skype and Microsoft PCs trace data sets.*

	Skype Trace	Microsoft PCs Trace
Mean *TTF* (seconds)	55,125	55,125
Mean *Down Time* (seconds)	51,509	15,906
Average percentage of online nodes	33.15%	81.24%

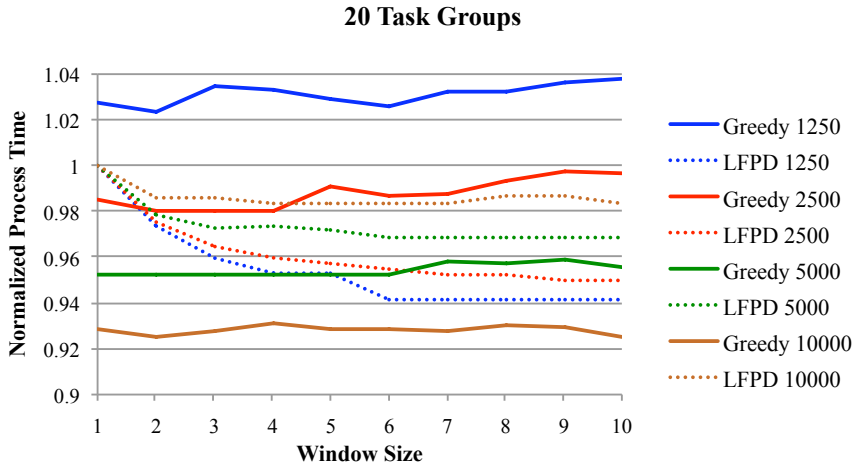
3.5.3 Performance Evaluation

The two dispatch policies are evaluated with different parameters and different resource availability trace data sets. The total process time of the computing job for different combinations are compared and discussed. To simply the discussion, all the results are normalized with the corresponding total process time of the simple redundant task dispatch policy.

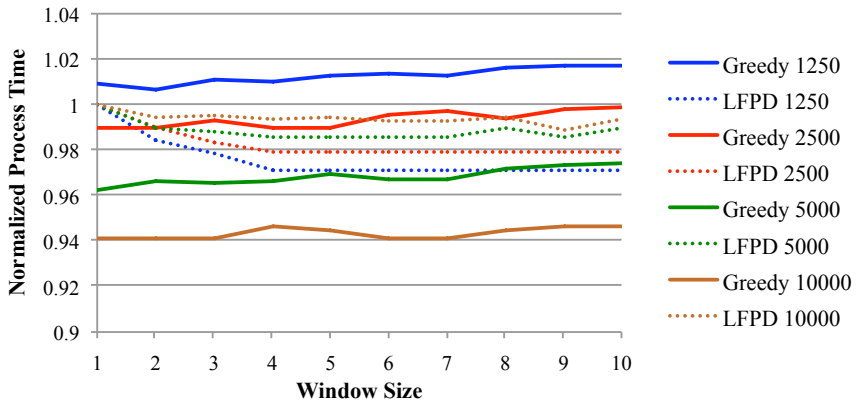
Figure 3.4 shows how the normalized total process time changes with the dispatch windows size and the mean task process time, using the Skype trace data set. The results with Microsoft PCs trace data set are shown in Figure 3.5.

Comparison with the Simple Redundant Task Dispatch Policy

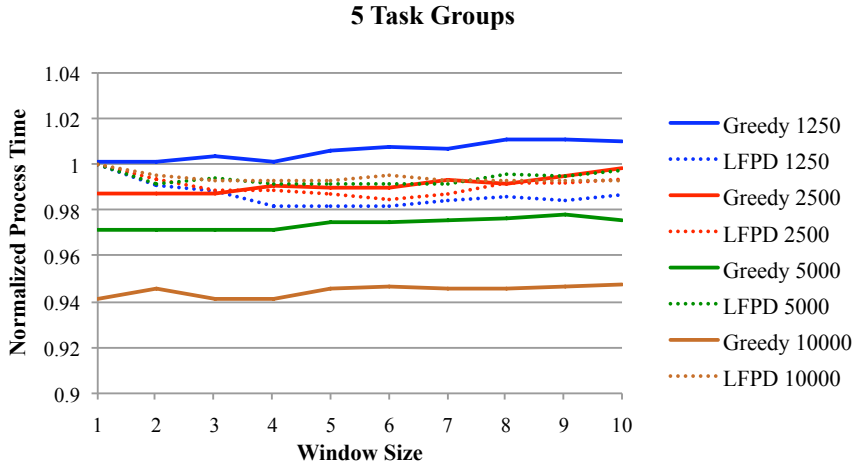
The results indicate that the *LFPD* policy outperforms the simple redundant task dispatch policy (window-size-1 *LFPD*). The improvement is more significant for a larger number of task groups. A smaller mean task process time also leads to a slightly better improvement. For 20 task groups and the mean task process time of 1250 seconds, *LFPD* delivers up



(a) 20 Task Groups.

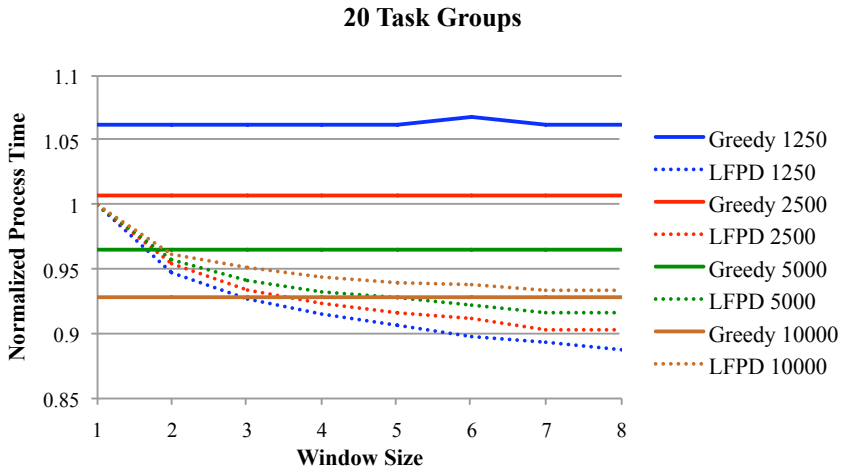


(b) 10 Task Groups.

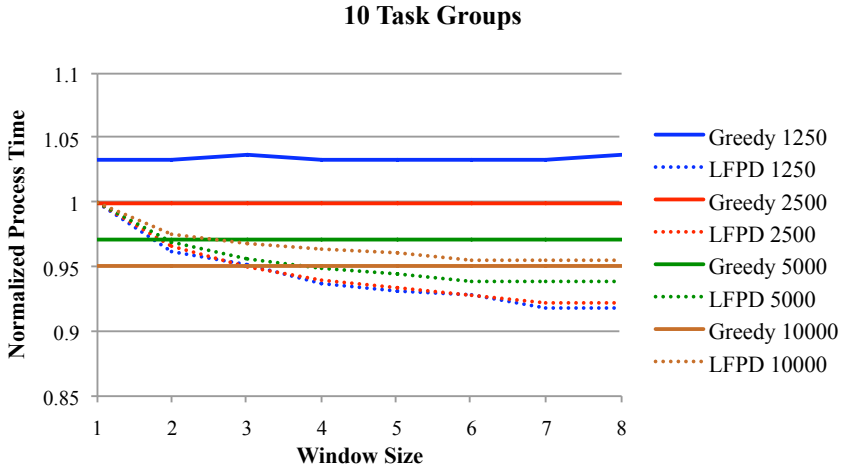


(c) 5 Task Groups.

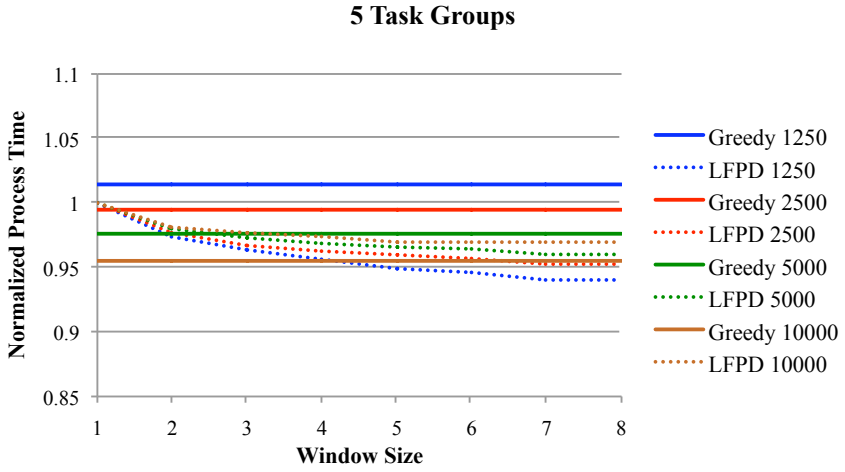
Figure 3.4: *Compare the* LFPD *policy and the greedy dispatch policy for different mean task process time (Skype Trace).*



(a) 20 Task Groups.



(b) 10 Task Groups.



(c) 5 Task Groups.

Figure 3.5: *Compare the* LFPD *policy and the greedy dispatch policy for different mean task process time (Microsoft PCs Trace).*

to 6% and 12% improvements for the Skype trace data set and Microsoft PCs trace data set, respectively.

The number of task groups is related to how many times a workflow is blocked during the process of a workflow. The "blocked" status introduces a serious performance overhead, because the computing power is used for re-execution of the failed tasks. The *LFPD* policy reduces the number of task failures, and thus mitigates this performance overhead. It explains why the *LFPD* policy is more efficient for larger number of task groups.

For a given trace data set, a larger mean task process time leaves less rooms for the *LFPD* dispatch to find a better assignment. As shown in Equation (3.4), the EFP of any task-to-worker assignment depends on the task process time, the current time, and the start time of the worker. Because the latter two values are given while finding a better task assignment, the EFP is only decided by the task process time. For example, there are two workers in the dispatch window, and two selected tasks. Given any assignment, a larger mean task process time leads to a longer task process time on both workers. Therefore, the EFP increases for both the two tasks. This EFP increment makes the overall failure probability (OFP) of both two possible assignments higher. The *LFPD* policy is designed to reduce the number of failures, by finding proper task-to-worker assignments. However, if all the assignments provide a high overall failure probability, the *LFPD* policy becomes less efficient. As a result, the *LFPD* policy delivers less improvement in the case of a larger mean task process time. In both of these two trace data sets, the mean TTF is 55125 seconds. The large mean task process time (10000 seconds) enlarges the tasks failure probability. Thus, the *LFPD* dispatch is less efficient, compared to the ones with a small mean task process time.

The results with two trace data sets are slightly different for their different availability characteristics. It is because of an overhead introduced by the dispatch window. When the dispatch window is not full, the workers that are waiting in the window are idle. Their computing power is wasted. Thus, the less time to fill a dispatch window, the better performance can be achieved. In the case of these two trace data sets, the average

number of online workers in the Microsoft PCs trace is much larger. Thus the time of the Microsoft PCs trace data set to fill a dispatch window can be expected to be much shorter than that of the Skype data set. Therefore, the *LFPD* policy delivers a better performance improvement with the Microsoft PCs trace data set for all the parameter combinations.

Comparison with the Greedy Dispatch

As shown in both Figures 3.4 and 3.5, the greedy dispatch policy beats the *LFPD* policy for a large mean task process time (10000 seconds). The reason is that the greedy dispatch policy eliminates all the task failures with its perfect knowledge of the worker availability status. While both the simple redundant task dispatch policy and the *LFPD* policy suffer the inefficiency for the high failure probabilities, the greedy dispatch policy is not affected. Therefore, the normalized process time with the greedy dispatch policy decreases while increasing the mean task process time.

These two figures also show that the *LFPD* policy is more efficient than the greedy dispatch policy for a small mean task process time. It is because of the "sleep" message used in the greedy dispatch policy. The greedy dispatch policy lets a worker to sleep if it finds this dispatch to be a waste of computing power. It happens when the existing copy of task has a smaller EFT than this new copy. This mechanism boosts the performance in most cases. However, it also introduces a possible overhead, for letting workers sleep even when the workflow is no longer "blocked" and "undispatched" tasks are available. When the mean task process time is small, the failures occur less frequently. Therefore, the greedy dispatch policy's advantage for eliminating task failures becomes less significant. In such cases, this particular overhead becomes more obvious and leads to worse efficiency.

How the Window Size Affects the Process Time

As shown in both Figures 3.4 and 3.5, the larger window size results in a shorter process time for the *LFPD* policy in most cases. It is because that the *LFPD* policy is likely

to find a better task-to-worker assignment with a larger window size, especially for the smaller mean task process time. As discussed earlier, the smaller mean task process time results in less frequent failures. Thus, the *LFPD* policy has a higher probability to find an assignment with less failures.

The overhead introduced by the "blocked" status is not serious when the number of task groups is small. Thus, the improvement achieved with the *LFPD* policy is small. Therefore, while the window size increases, the overhead for the dispatch window becomes obvious. The overhead is more serious when the number of online workers is small. It explains why the process time with the *LFPD* increases when the window size exceeds a certain value in Figures 3.4(b) and 3.4(c). With a much larger number of online workers, the overhead for the dispatch window is not significant. Thus, the results with the Microsoft PCs trace are not affected by the small number of task groups and a large window size.

Improve the Performance by Identifying Worker Types

In the real world, multiple types of workers exist. Different type of workers has different availability characteristics. Nadeem et al. [53] introduced the class level modeling method by pre-identifying three types of resources in the Austrian Grid [60]. The *TTF* distribution of different types of resources is largely different across the three types. The heuristics-based failure estimation relies on the empirical distribution, and assumes that all the workers have similar availability behavior. Gathering multiple type of workers *TTF* into single *TTF* distribution leads to low estimation accuracy. The low estimation accuracy will affect the performance, because the *LFPD* policy cannot find the optimal task-to-worker assignments with the inaccurate failure estimations.

To improve the failure estimation accuracy, the worker type is considered. Two types of workers are selected from the two real world trace data sets. First, the two trace data sets are clustered into several types, using a K-Means clustering algorithm in the Weka toolkit [67]. By extracting the *TTF* and the down time pair from the original

Table 3.3: *Clustering results* (node distribution, mean uptime/mean downtime).

	Microsoft	Skype
Cluster1	18.4%, 20.66 days/13.97 hrs	4.0%, 8 days/6.68 hrs
Cluster2	5.7%, 28.68 hrs/4 days	6.3%, 9.49 hrs/4.40 days
Cluster3	62.2%, 16.1 hrs/6.76 hrs	79.6%, 6.73 hrs/8.72 hrs
Cluster4	13.7%, 7.97 days/8.99 hrs	10.1%, 2.75 days/6.50 hrs

trace data sets, two dimensional data are generated. The number of clusters is four, based on the assumption that four kinds of workers (diurnal, weekly, long *TTF*, and long downtime) exist. The clustering results are shown in Table 3.3. Each cluster shows different characteristics. Cluster 3 of Microsoft PCs trace shows a diurnal pattern, while Cluster 3 of Skype trace is highly volatile.

The same number (5,000) of peers are selected from the major cluster of both trace data sets to form a new 2-type trace data set. Thus, this 2-type trace data set is considered to consist of two types of workers. This trace data set is used in the simulation to study the effect of identifying worker types.

The parameters of the simulator are listed in Table 3.4. The *LFPD* policy with and without the ability to identify two worker types are simulated. If two types of workers are identified, each type of worker's *TTF* data is gathered separately. When a failure probability estimation is needed for a task-to-worker assignment, the dispatcher selects the corresponding *TTF* distribution for each worker and then estimates the failure probability of the worker.

Figure 3.6 shows improvement achieved by identifying the worker types. The y-axis represents the normalized total process time with identifying the worker types. These results are normalized by the total process time without identifying the worker types. The results with the ability to identify worker types show an average improvement of 0.7% (ranges from 0.1% to 1.5%). The results also indicate that the improvement is more significant for a larger mean task process time, a larger number of task groups, and a

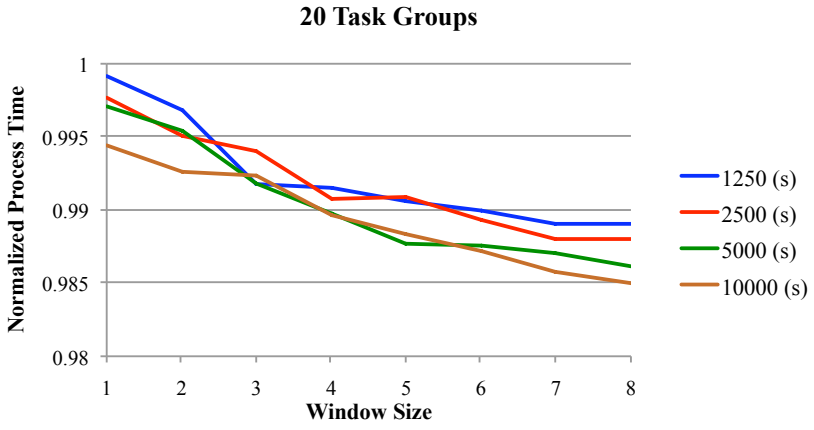
larger dispatch window size. As discussed in Section 3.5.3, a larger mean task process time leaves less rooms for the *LFPD* dispatch to find a better assignment. Therefore, the accuracy of failure estimation has a bigger impact on the performance. It has also been discussed that a larger number of task groups makes the performance degradation more serious. Thus, accurate failure estimation offers a higher improvement. The larger the dispatch window is, the more possible task-to-worker assignments exist. If failure estimation is not accurate, the *LFPD* policy cannot find the optimal assignment from these assignments. Therefore, the accuracy of the failure estimation is more critical.

Table 3.4: *Simulation parameters for the 2-type trace data set.*

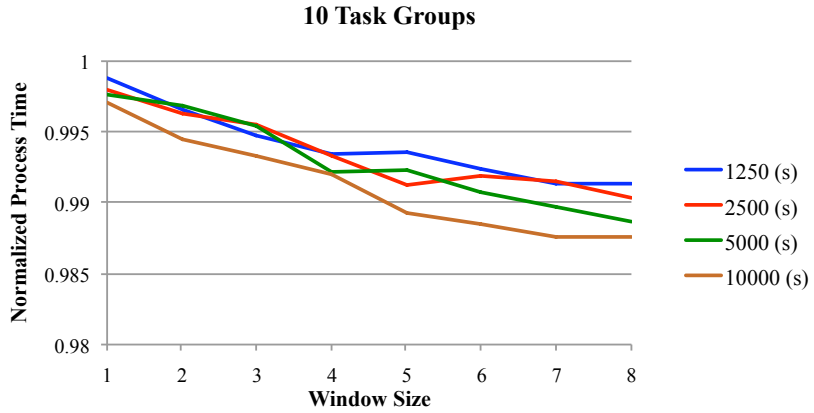
	2-type Trace Data
Number of Workers	5,000 (Volatile) + 5,000 (Diurnal)
Number of Tasks	400,000
Mean Task Process Time	1250, 2500, 5000, 10000 (s)
Number of Task Groups	5, 10, 20
Window Size	1, 2, 3, 4, 5, 6, 7, 8
Idle Worker Inquire Interval	200 seconds

3.6 Conclusions

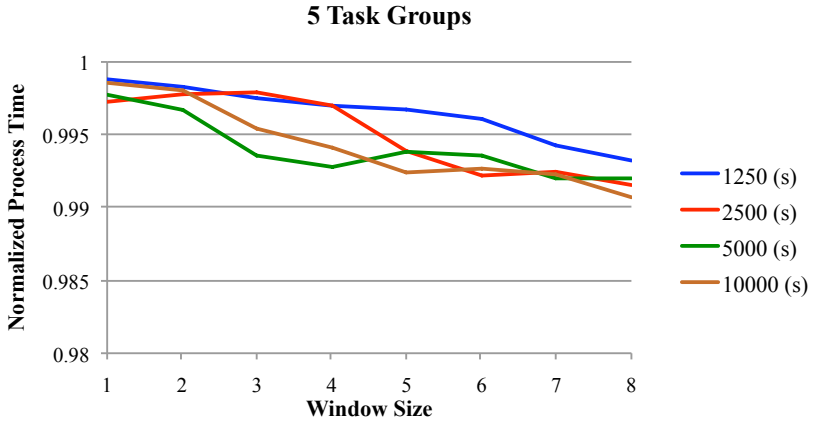
The redundant task dispatch policy proposed in Chapter 2 has a major limitation: the average failure rate model is not the best fitted for the volunteer peers in the real world. To address this limitation, this chapter has proposed a heuristics-based mechanism for failure probability estimation based on the life cycle model of volunteer peers and the statistical data. Then, the *Least Failure Probability Dispatch (LFPD)* policy has been introduced. Instead of dispatching a task when a task enquiry comes, this dispatch policy waits for several task enquiries from different workers, and then dispatch tasks to them together. It uses the heuristics-based failure probability estimation method to find an optimized task-to-worker assignment that minimized the overall failure probability of the



(a) 20 Task Groups.



(b) 10 Task Groups.



(c) 5 Task Groups.

Figure 3.6: *The improvement archived by identifying multiple worker types.*

tasks. The *LFPD* policy has been evaluated with real world trace data sets on a simulator. The evaluation results have been compared with those of two selected baseline policies. The comparison results indicate the effectiveness of the *LFPD* policy. The results also prove that the *LFPD* policy can beat the greedy dispatch policy when the mean task process time is much smaller than the mean TTF of the workers. The difference between the results with two trace data sets is also discussed. To study how the different type of workers in the real world may affect the effectiveness of the *LFPD*, a trace data set that consists of two types of workers has been generated from the two real world trace data sets. The *LFPD* policy has been simulated with and without the ability to identify different type of workers. The results indicate that worker type identification can provide additional performance improvement.

Chapter 4

Hardware-Based Secure Volunteer Data Processing

4.1 Introduction

Human beings are producing and gathering rapidly increasing of data (text, image, video etc) volume these years. Processing these data requires massive computing power. While major volunteer computing platforms have been used to process publicly available data, there are many other types of data processing that cannot utilize their massive computing power. To process sensitive data such as medical data (patient data etc), biology data (DNA etc), market research data (customer data etc), financial data (bank account etc), a volunteer computing platform needs security features that can protect the data on the anonymous volunteer peers.

This chapter explores the potential of secure data processing on the volunteer computing systems using a hardware-based cryptography approach. The related work of privacy preserving data mining [68] is first discussed. Then, a general secure data processing method for the volunteer computing is proposed. While the encryption of the data owner's server can be trusted, the decryption of the volunteer workers is challenging. The decryption on the workers can be implemented using software-based and hardware-

based approaches. The advantage of the hardware-based approach is discussed. A sample application and a widely available processor with hardware security features are used to assess this hardware-based secure data processing method. The application is intensively optimized for the architecture of this secure processor to archive high performance secure data processing. The evaluation results indicate that a secure data processing application on a secure processor outperforms a non-secure data processing application running on commodity processors, but also demonstrates a considerable performance overhead introduced to achieve the hardware-based secure data processing.

4.2 Related Work

The security method in this work is related to a novel research direction in data mining - privacy preserving data mining [68]. The main objective in privacy preserving data mining is to develop algorithms for modifying the original data in some way, so that the private data and private knowledge of any participants remain private even after the mining process. The typical modification methods include: perturbation, blocking, aggregation/merging, swapping, and sampling. A number of algorithms have been designed for different data mining techniques, such as classification, association rule discovery, and clustering. These algorithms can be classified into the following three types:

- Heuristic-based: has side effects caused by selective data modification or sanitization.
- Cryptography-based: conducts data mining on private data from multiple parties. None of these parties is willing to disclose its own data and found knowledge. It is referred to as the Secure Multiparty Computation (SMC) problem.
- Reconstruction-based: perturbs the data and reconstructs the distribution at an aggregate level.

For all the above three types, specific algorithms are required to handle the privacy issues for different data mining techniques. For example, numbers of specific algorithms have been proposed for cryptography-based association [69,70], decision tree induction [71, 72], and clustering [73]. The heuristic-based [74, 75] and reconstruction-based algorithms [76–79] introduce side effects on the found knowledge. The cryptography-based algorithms can only apply to the SMC problems which is not suitable for volunteer computing, because of the different ownership of sensitive data.

To process sensitive data on a volunteer computing platform, a privacy preserving technique is required to guarantee that the owner of a volunteer peer cannot access sensitive data and the knowledge inside the data.

4.3 Hardware-Based Secure Data Processing

A cryptography-based method that encrypts sensitive data on the data owner's server and decrypts the data on the workers is proposed in this chapter. This method makes it possible to process sensitive data without disclosing the sensitive data, while no side effect is introduced.

4.3.1 Problem Definition

On the volunteer computing platform, volunteer peers process tasks dispatched from a dispatcher. Volunteer peers do not own the data inside the tasks that are dispatched to them. Therefore, the problem here is how to insure that any unauthorized access to the plaintext data of computing tasks is disabled on the volunteer peers to guarantee the data security.

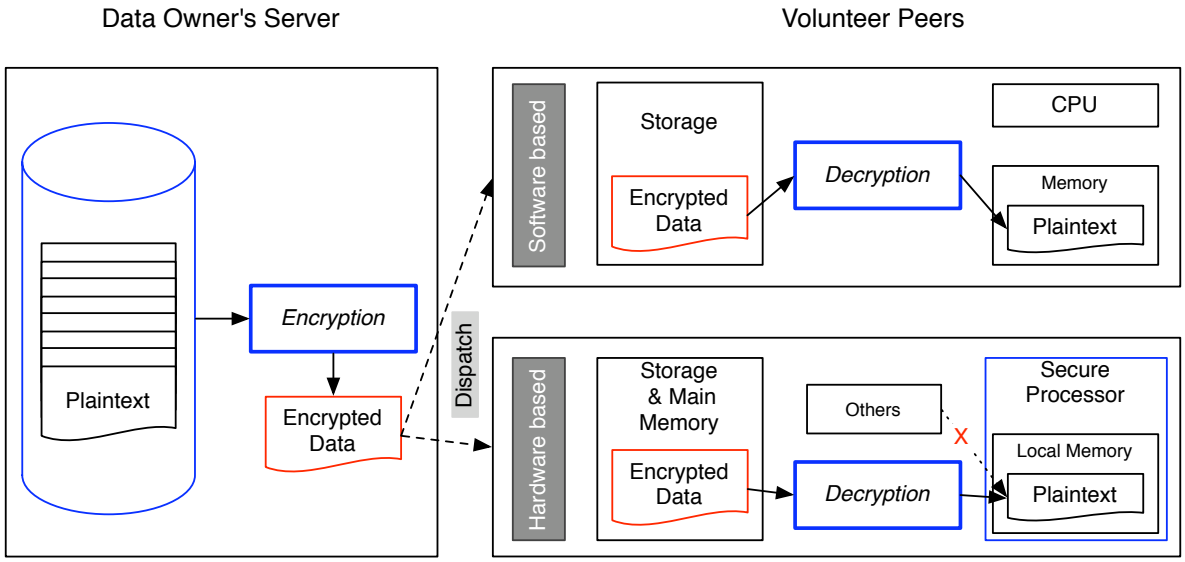


Figure 4.1: *Data transfer flow and the cryptography process.*

4.3.2 The Secure Data Processing Method

The secure data processing method relies on a basic cryptography method. The only two places where plaintext data may exist are the data owner's server and a special memory space of a volunteer peer that cannot be tampered even by the peer owner.

As shown in Figure 4.1, before dispatching to the volunteer peers, plaintext data are encrypted by the data owner's server. The encrypted data are dispatched to the volunteer peers. The volunteer peers decrypt the data with a proper application key and process the plaintext data. By insuring the safety of the application key and denying the access to the plaintext data in memory, the sensitive data are protected on the volunteer peers.

There are two approaches to achieve the data protection on the volunteer peers. One is the software-based approach, which assumes that the operating system and the hardware system are safe. As shown in Figure 4.1, the encrypted data are decrypted with the application key and stored in the main memory. This approach is vulnerable for the following reasons:

- Operating system could be hijacked.

- Hardware such as main memory is vulnerable to physical attacks.
- The volunteer peers' owners have full access to their peers. It makes the situation worse, because they can try to access the data on purpose.

The other is the hardware-based approach, which trusts neither the operating system nor the main memory. The only trusted entity is a secure processor [80] and the local memory inside this processor. Such a secure processor has an embedded hardware key. The key hierarchy is built on top of this key. The application key is embedded in the task application image and authorized by the data owner. The application image is also encrypted with a key in the key hierarchy, and can only be decrypted to run on the secure processor. Thus, the application key inside the application image is not accessible from anywhere outside the secure processor. Therefore, the encrypted data cannot be decrypted on any volunteer peers other than an secure processor running an authorized computing task. By insuring that the plaintext data in Figure 4.1 are only accessible for the data owner and the computing application authorized by the data owner, the data security is guaranteed.

4.4 Case Study of a Secure Processor - The Cell Processor

To explore the potential of using the hardware-based approach to process sensitive data on the volunteer computing platforms, a secure processor - the Cell processor that is built into millions of volunteer computing peers (PlayStation3) is first analyzed.

4.4.1 Cell Architecture Overview

As shown in Figure 4.2, the Cell processor is a single-chip multiprocessor with nine cores [81,82]. The nine cores, main memory and I/O are connected via the Element Interconnect

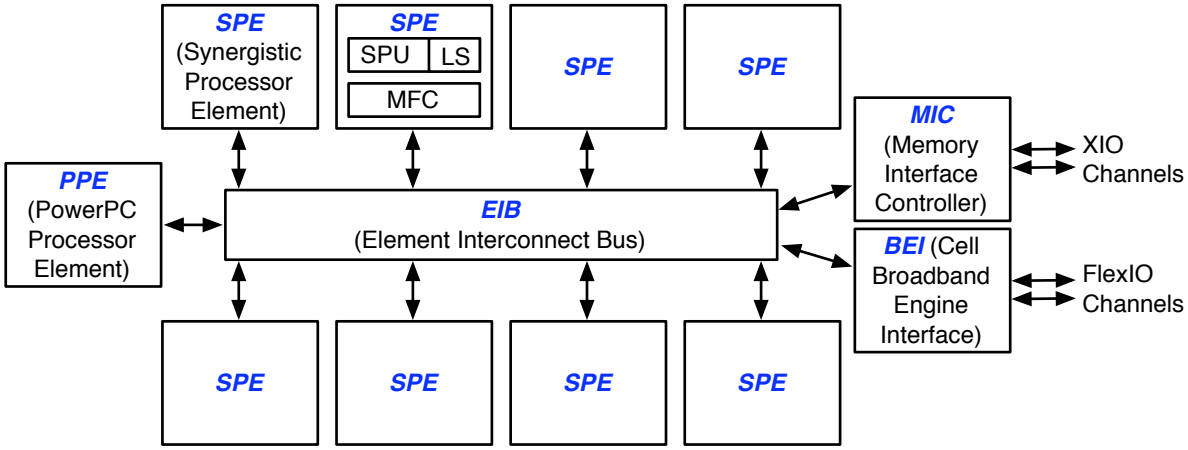


Figure 4.2: *Cell architecture overview.*

Bus (EIB). One of the cores, the PowerPC Processor Element (PPE) is responsible for overall control of the system. The Synergistic Processor Elements (SPEs) are 128-bit SIMD cores optimized for data-rich operations. The PPE allocates computation-intensive applications to the SPEs for processing. Each SPE contains a RISC core called Synergistic Processing Unit (SPU), 256KB software-controlled Local Store (LS), and a Memory Flow Controller (MFC) that controls direct memory access (DMA) for data transfer between the main memory and the LS. Each SPU can only access its own LS directly.

Data transfer between the main memory and the LS is handled by software-controlled DMA operations. The SPU can execute instructions while the MFC processes the DMA operations concurrently. Thus, double-buffered DMA transfer can be utilized to overlap computation and DMA transfer.

The Cell processor's peak single precision performance is 25.6Gflop/s@3.2GHz per SPE. Its peak double precision performance (1.83Gflop/s@3.2GHz per SPE) is not as high as the single precision one, but still impressive.

4.4.2 Programming Models

As shown in Figure 4.3, the data parallel model and the pipeline model are two simple programming models [83] for the Cell processor. In the pipeline model, an application is

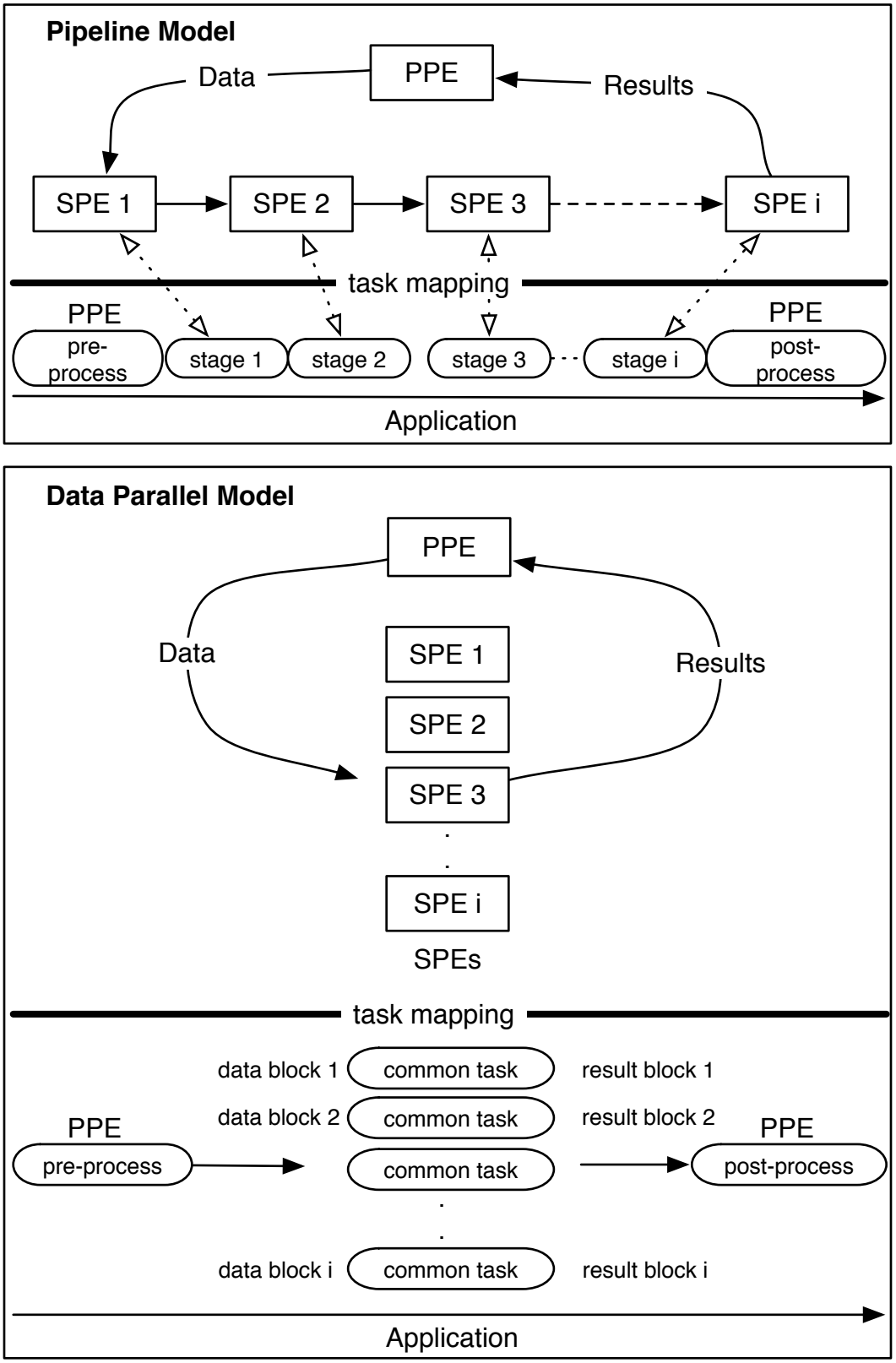


Figure 4.3: *Basic programming models for the Cell processor.*

divided into several stages. Each stage is mapped to a task, which will be running on one of the SPEs. Intermediate results are passed from one SPE to the next for processing. In the data parallel model, the PPE acts as a controller. A common task is allocated to the SPEs. A data set is divided into blocks for processing. Each SPE processes a different data block. In this work, the data parallel model is used, which is best suited for most data mining algorithms that involve massive data parallelism.

4.4.3 Cell Security Features

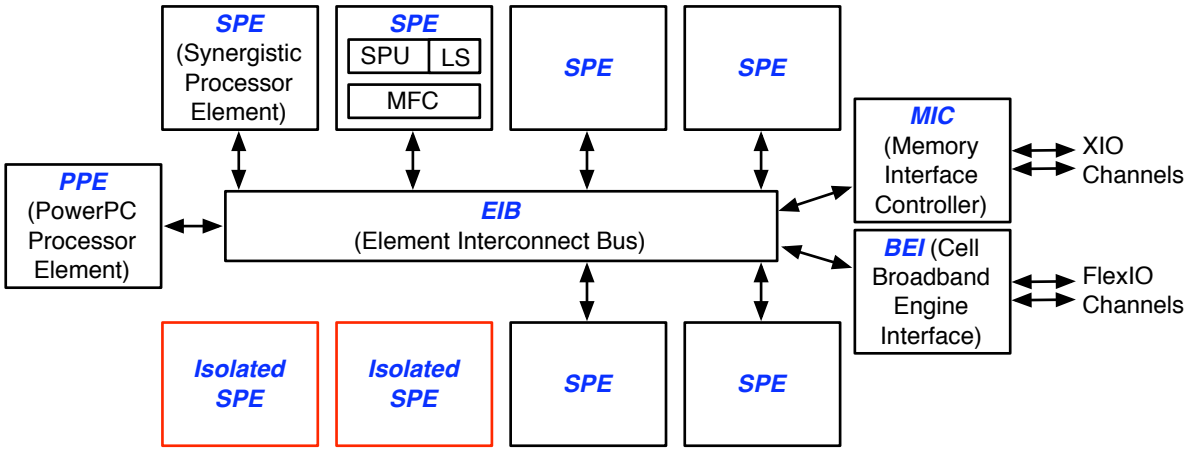


Figure 4.4: *Isolation mode of the Cell processor.*

Besides the performance, Cell is also compelling because of the hardware security features. The Cell processor comes with the hardware security features proposed in [80]. The core of Cell's security features is the isolation mode of SPE [84, 85]. As shown in Figure 4.4, by isolating a SPE, its LS is locked for its own use. A small area of the LS is left open for communication purposes. External execution path control of the SPE is also disabled. Thus, the only possible external action for an isolated SPE is "cancel." When an isolated SPE is cancelled, all the data in the LS and registers are erased before external access is enabled. The basic functionality provided by the current security SDK includes: the "decrypt-in" function and the "encrypt-out" function. The former function decrypts encrypted data with a user defined 128-bit application key and puts the decrypted

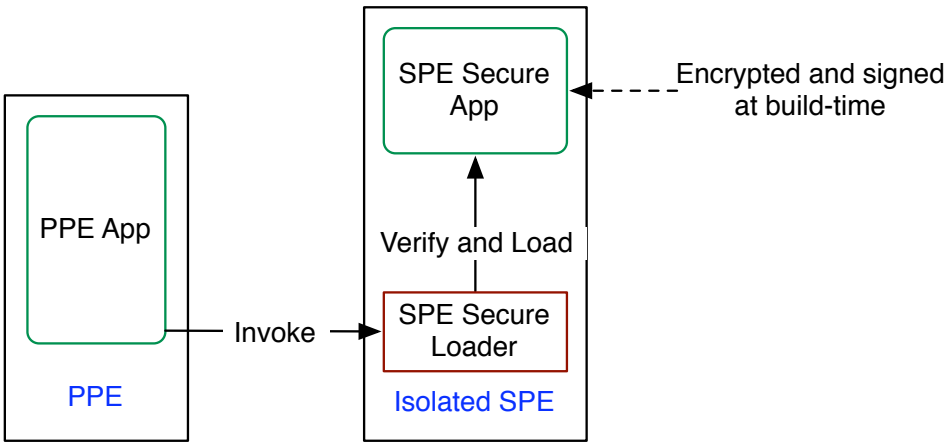


Figure 4.5: *Invoke an isolated SPE thread.*

plaintext data in the LS. The latter one does the work on the opposite direction. The application key is protected through the following key hierarchy on top of the Cell security architecture.

The Cell processor has an embedded hardware root key. Any computing task running on an isolated SPE needs to go through a trust chain starting from the hardware root key before execution.

As shown in Figure 4.5, when an isolated SPE thread is invoked by the PPE, the SPE secure loader is loaded on the SPE. The SPE secure loader has an embedded public key as the root certificate authority (CA). The hardware root key is used to verify the signature on this public key. If the verification fails, the root CA of this SPE secure loader might have been tampered, and thus the execution stops. Otherwise, the SPE secure loader starts and goes through several level CA public key verifications until the first level application authentication key.

For building a secure SPE application, the application is encrypted with derivatives of two keys: the public key of the SPE secure loader's application decryption key pair, and an application authentication key owned by the developer. The private key of the key pair is embedded in the SPE secure loader. The application authentication key is embedded into the encrypted application image with the signature value. Because of the derivative,

an application cannot decrypt the encrypted section of another application developed by other developers.

At the execution, the SPE secure loader first verifies the application image with the embedded application authentication key. Then, the secure loader decrypts the image with the derivative of the two keys and starts this application in the isolation mode.

By running a computing task on an isolated SPE, any access from outside of this SPE to the SPE's LS is prohibited. Thus, to preserve the privacy of data, data can be encrypted by its owner with an application key. No one can decrypt the data without the appropriate application key. This application key is protected by the application authentication key of the data owner, and eventually protected by the hardware root key. Only the decrypted computing task running on the isolated SPE can decrypt and access the data with its embedded key.

The correctness of the Cell processors security features, and details on key hierarchy and API of the security features can be found in [84, 85].

4.5 Sample Application: Secure K-Means Clustering

Applying the proposed hardware-based security method to the Cell processor, secure data processing on volunteer computing platforms can be implemented. To explore the potential of this method, a classic data clustering algorithm, so-called K-Means algorithm [86], is used as a sample application to study the performance impact of the security feature. First, a non-secure parallelized K-Means algorithm for the Cell processor is designed to fully utilize the computing power of the Cell processor. Then, a secure K-Means algorithm is designed on top of that, together with the secure data processing method.

4.5.1 K-Means Algorithm

K-Means clustering is one of the simplest learning algorithms for data clustering. As shown in Figure 4.6, the procedure follows a simple and easy way to cluster a data set into a certain number of subsets.

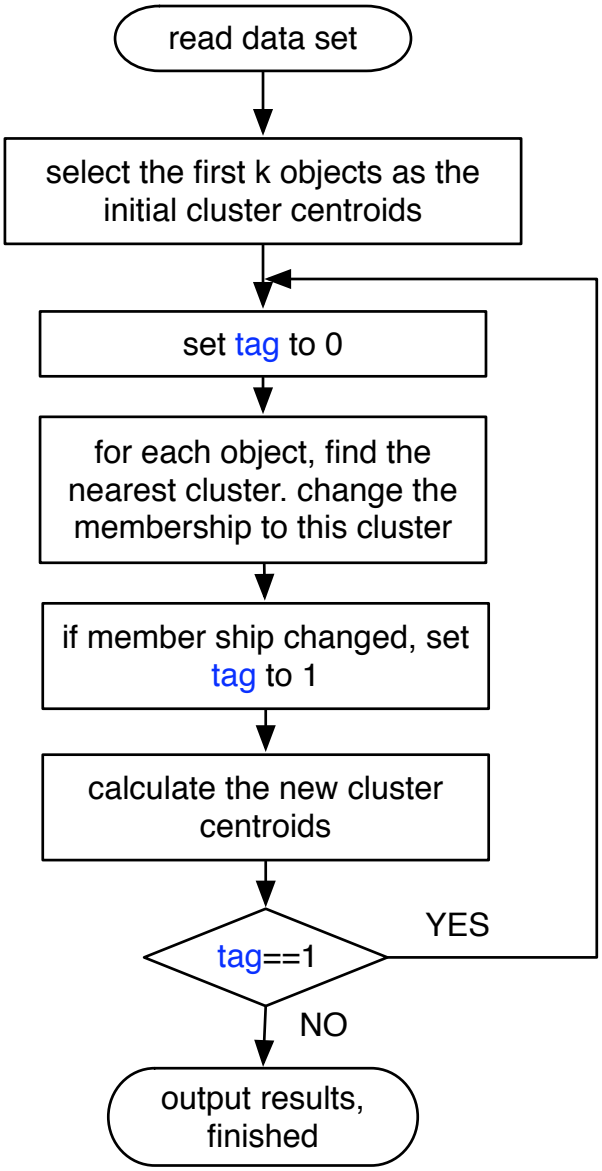


Figure 4.6: *K-Means algorithm.*

Let k be the number of clusters. In the first step of the procedure shown in Figure 4.6, k centroids are selected from the data set, one for each cluster. The next step is the

beginning of a loop, which initializes "tag" to 0. The second step of the loop is to associate each object of the data set to the nearest centroid. Thirdly, k new centroids are calculated. After that, a new binding has to be done between the data set objects and the nearest new centroid. If any object's membership changes, "tag" is set to 1. Then, the new cluster centroids are calculated. The last step in the loop is the termination test of the "tag" value. If the "tag" value is 0, the loop is terminated and the post-process starts.

As a result of this loop, the k centroids change their locations step by step until no more changes are done. In other words, the centroids do not move any more if none of the object's membership changes. The algorithm iteratively reduces an objective function:

$$\sum_{i=1}^{k} \sum_{n \in = S_i} |x_n - c_i|^2, \tag{4.1}$$

where S_i is a cluster, and $|x_n - c_i|^2$ is the distance between a data object x_n and the cluster centroid c_i. The function is minimized by iterating the loop described in Figure 4.6.

4.5.2 Parallelized K-Means Clustering for The Cell Processor

A parallelized K-Means algorithm is designed based on the data parallel model shown in Figure 4.7. As shown in Figure 4.7, the process is divided into two parts that are executed by the PPE and the SPEs, respectively. The PPE is designed to be responsible for overall control of the system. The SPE's architecture is optimized for computation-intensive applications. K-Means clustering involves massive data-parallelism, which is suitable for the SPE's 128-bit SIMD operations. Thus, most of computation is mapped onto the SPEs to utilize their computing power. The PPE only calculates the new centroids with the local results from the SPEs.

In the Cell processor, a communication mechanism between the PPE and SPEs is called mailbox. Two mailboxes (the *SPU write outbound mailbox* and the *SPU write*

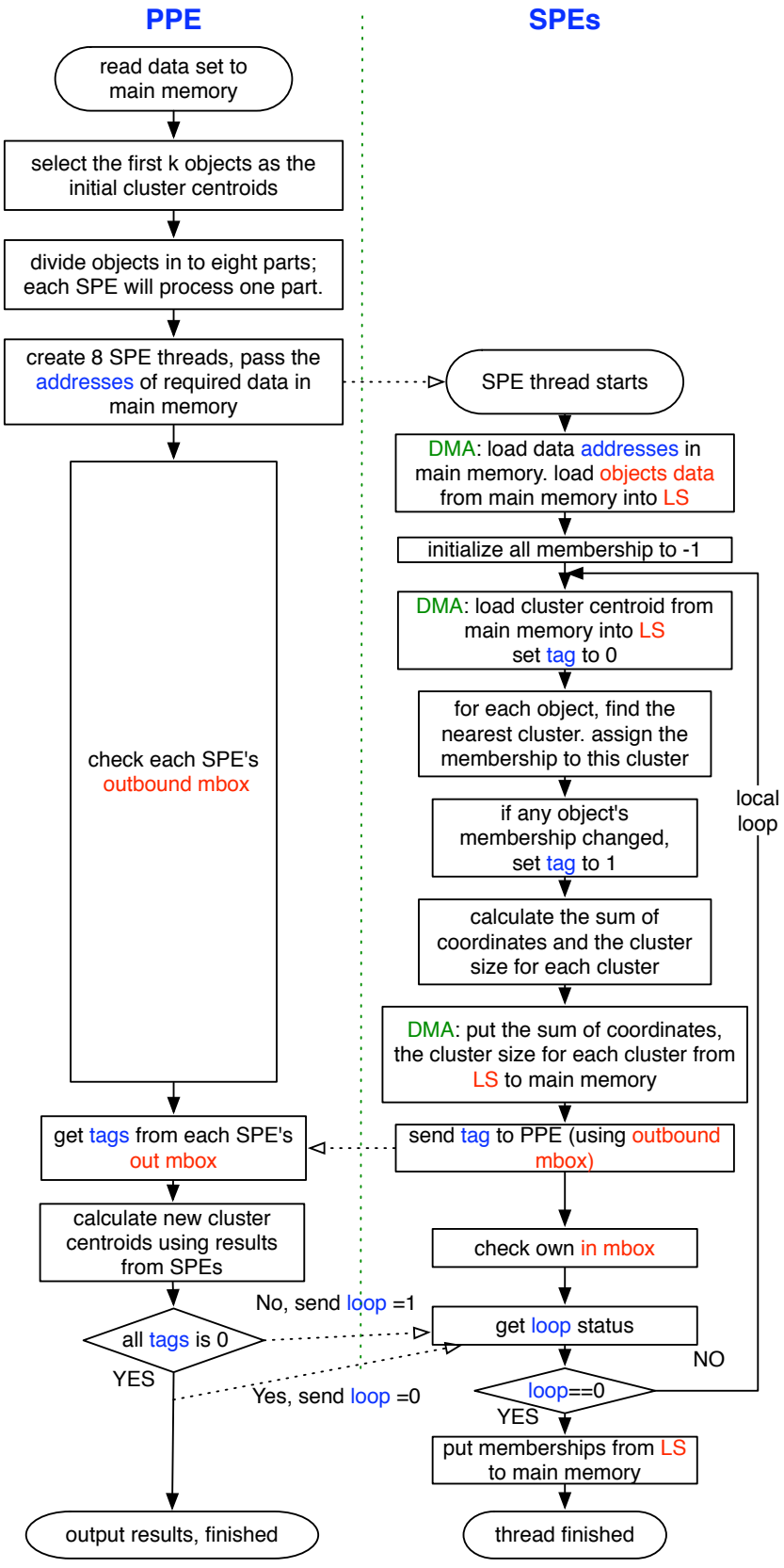


Figure 4.7: *K-Means algorithm for the Cell processor.*

outbound interrupt mailbox) are provided for sending messages from the SPE to the PPE. One mailbox (the *SPU read inbound mailbox*) is provided for sending messages to the SPE.

The details on the processing diagram are as follows:

PPE: As shown in Figure 4.7, the PPE first reads the data set into the main memory and selects first k objects as centroids, one for each cluster. The data set is separated into several parts. Each part will be allocated to one SPE. Then, the PPE creates SPE threads and passes the address of required data in the main memory to each SPE. Then the PPE will keep checking the SPU write outbound mailboxes.

The next step on the PPE starts after getting all the "tags" from the SPU write outbound mailboxes, the PPE calculates new cluster centroids using the results from the SPEs, and checks if all the "tags" equal 0. If all the "tags" equal 0, the K-Means clustering is finished. Then, the PPE communicates with the SPEs using the the SPU read inbound mailboxes to finalize the SPE threads. Otherwise, the PPE communicates with the SPEs to start the next iteration of each SPE's local loop.

SPE: After a SPE thread is started, it first loads the required addresses in the main memory. The address of these required addresses is passed to the SPE when the PPE start the SPE threads. Then, a part of the object data are loaded into the SPE's LS with the address in the main memory, using DMA operations. After the DMA transfer is finished, the membership of every object on this SPE is initialized to "0."

The SPE's local loop is as follows. The first step is the membership assignment. If any object's membership changes, the "tag" is set to "1." The SPE thread calculates the sum of coordinates and the cluster size for each cluster, and then transfers these data back to the main memory using a DMA transfer. When this transfer is finished, the SPE puts "tag" to its SPU write outbound mailbox and starts checking its SPU read inbound mailbox.

To fully exploit the computing power of the Cell processor and achieve high performance volunteer data processing, several challenges have to be considered: the software-

controlled memory hierarchy, the parallelism among the SPEs, and the 128-bit SIMD ISA of the SPE. Software controlled double-buffered DMA transfer can mitigate memory latency. Parallelism among SPEs is naturally achieved in the data parallel model. Because the focus of this work is on the Cell's specialized architecture and performance for data clustering, the efficacy of the compilers for auto-SIMDization and loop unrolling is not explored. Instead, it relies on SIMD intrinsics provided by SPU C/C++ Language Extensions (Intrinsics) and hand-tuned loop unrolling.

The optimizations for the K-Means algorithm include:

- Use SIMD intrinsics in SPE threads to exploit the data parallelism.
- Overlap computation and DMA transfer by double buffering to reduce DMA stalls.
- Unroll loops to provide better dual issue rates.

SIMDization

The SPU ISA operates primarily on SIMD vector operands, both fixed-point and floating-point values. The SPE programming model defines 11 vector data types, including *vector float* and *vector double*[1]. The vector data types are all 128-bit long, containing four single precision floats or two double precision floats. SPU C/C++ Language Extensions provide a convenient interface for the SPU ISA and hardware features to C programmers. Intrinsics *spu_sub(a,b)* and *spu_madd(a,b,c)* will be used in the parallelized K-means implementation.

The core computation in a SPE thread is calculation of the distance between an object and a cluster centroid. The scalar code is shown as follows. Inside the loop, each iteration calculates the distance on one dimension. Because there is no dependency among the calculation of the distance on different dimensions, the distance calculation on different dimensions can be processed in parallel. Thus, the distance calculation can be using SIMD intrinsics.

[1]PPU instruction set does not support double-precision floating point vector.

```
1 // Scalar code
2 dist=0;
3 for(j=0; j<dim; j++)
4   dist += (objs[i*dim+j]-clusts[j]) * (objs[i*dim+j]-clusts[j]);
```

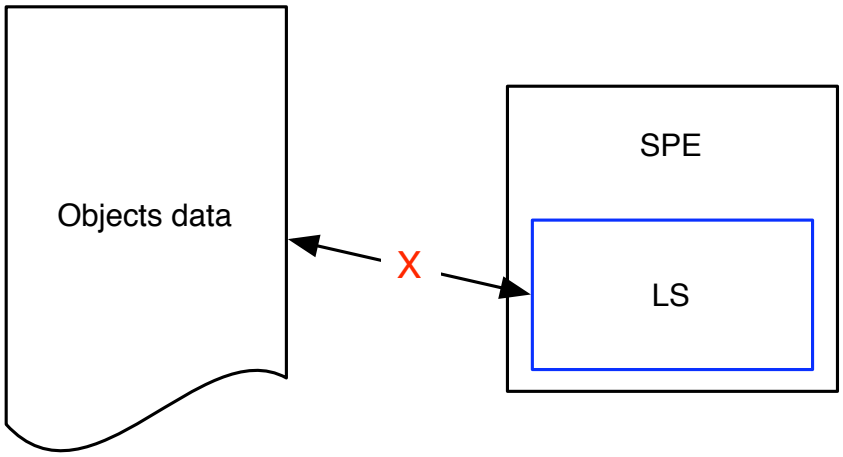
Using single precision floating point type as an example, the SIMD code is shown as follows. Inside the loop, the data type is *vector float*, which contains data of four dimensions. Each iteration calculates the distance on four dimensions.

```
1 // SIMD code
2 dim_vec = (dim+3)/4;
3 vec_dist = (vector float)(0.0,0.0,0.0,0.0);
4 for(j=0; j<dim_vec; j++) {
5   obj = objs[buffer][i*dim_vec+j];
6   local_dist = spu_sub(obj,clust[j]);
7   vec_dist = spu_madd(local_dist,local_dist,vec_dist);
8 }
```

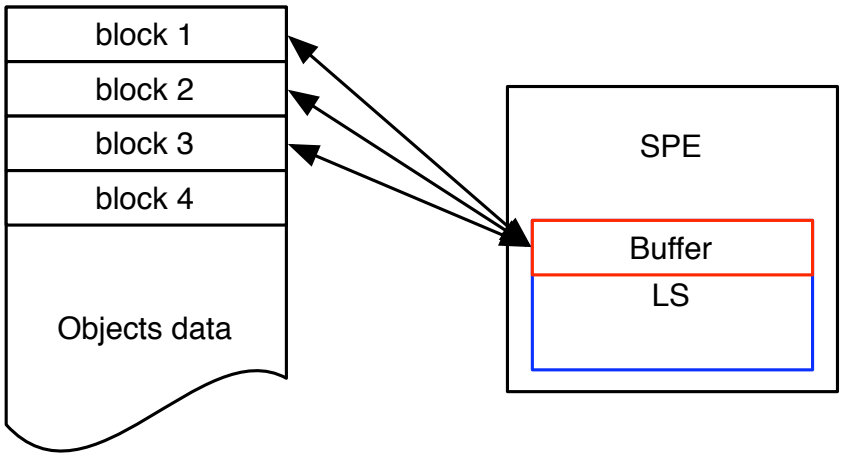
Double Buffering

As shown in Figure 4.8(a), each SPE has a 256KB LS. When the object data mapped to a SPE can be larger than 256KB, the object data cannot fit within the LS. In this case, data have to be divided into data blocks for processing on the SPE. Therefore, a data buffer in LS is required. As shown in Figure 4.8(b), the process of each data block includes the following three steps:

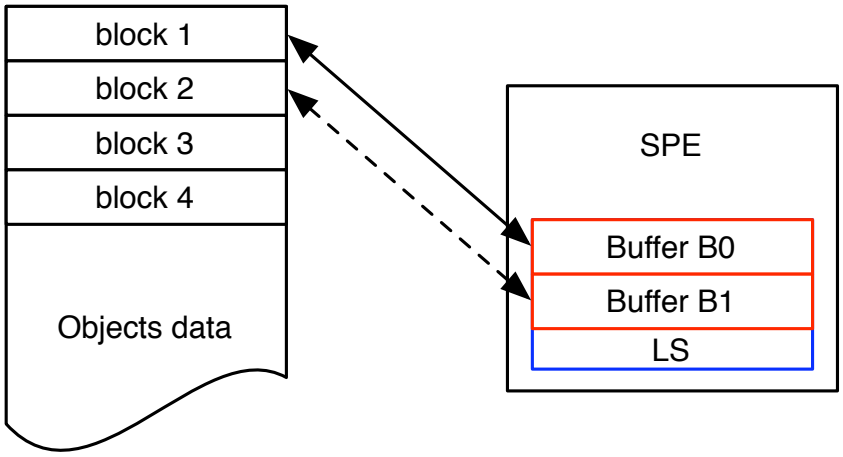
1. Get block i from the main memory to a buffer.
2. Process data in the buffer.
3. Put data in the buffer back to block i.
4. Increase i by one.



(a) Data cannot fit within the LS.



(b) Using data buffer to load data blocks.



(c) Using double buffering to overlap computation and data transfer.

Figure 4.8: *LS and buffering.*

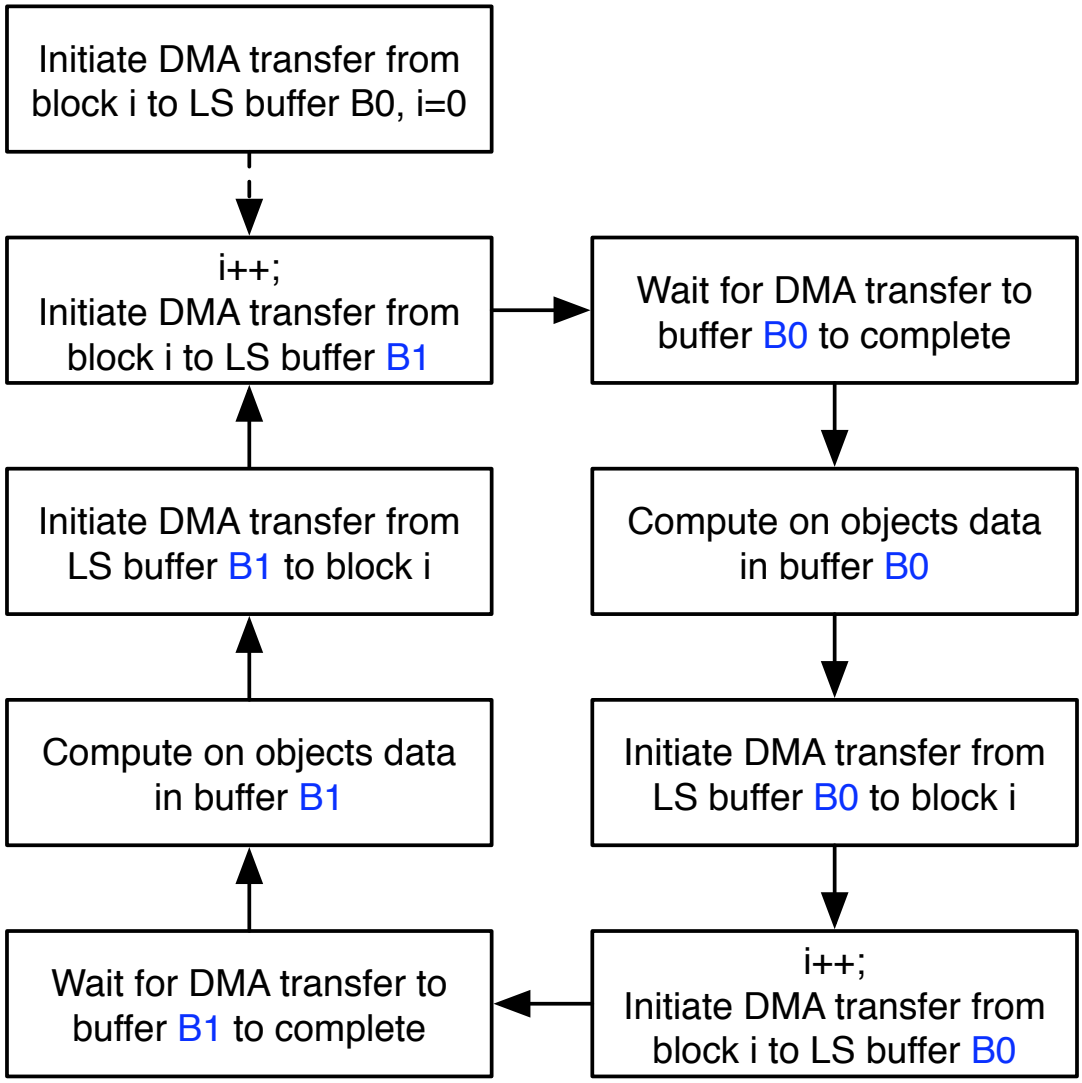


Figure 4.9: *The double buffering scheme.*

In such a case, a great deal of time is wasted for waiting for DMA transfers to complete. The SPU can execute instructions while the MFC processes the DMA commands concurrently. Thus, double-buffered DMA transfer can be used to overlap computation and DMA transfer. As shown in Figure 4.8(c), two buffers, *B0* and *B1*, are allocated. The computation on one buffer and data transfer in the other are overlapped. Figure 4.9 shows a diagram for the double buffering scheme for the K-means algorithm.

Loop Unrolling

To improve the performance, the compiler needs more instructions to schedule, so that the program does not stall. Most compilers can automatically unroll loops. However, automatic loop unrolling is not always effective. Because the efficacy of the compilers is not discussed, hand-tuned loop unrolling is applied here.

For the K-means process on each SPE, the first level loop is the loop of K-means iterations. Because cluster centroids will change after each iteration, inter-loop dependency exists. The second level loop is the process of objects in a buffer. Since each iteration finds the membership of one different object, there is no inter-loop dependency. An example of loop unrolling for the second level loop is shown as follows. Each iteration finds the membership of four different objects. Thus, much more independent instructions are available to compiler for scheduling. Ultimately, the performance is improved.

```
1 // Unrolled loop
2 for (i=0; i<objPerBuffer; i+=4){
3   process(obj_buffer[i]);
4   process(obj_buffer[i+1]);
5   process(obj_buffer[i+2]);
6   process(obj_buffer[i+3]);
7 }
```

4.5.3 Secure K-Means Clustering

Applying the secure data processing method to the parallelized K-Means clustering algorithm, secure volunteer data clustering can be achieved. The overall process flow is similar to the non-secure algorithm, with extra data decryption while loading data from main memory.

The main difference between clustering processes on plaintext data and encrypted data is shown in Figure 4.10. While plaintext data that is transfered from main memory to SPE's LS requires only MFC to execute DMA commands, encrypted data decryption

occupies both MFC and SPU. The SPU handles all the decryption computation. Thus, unlike the non-secure parallelized K-Means clustering on the Cell processor, the secure K-Means clustering cannot take advantage of the double buffering optimization. The effect of the lack of double buffering on performance will be discussed in the next section. Other modifications on the parallelized K-Means clustering are related to the isolation mode activation and secure data transfer functions [85]. The IBM Cell BE Security SDK provides the *decrypt_in*() and *encrypt_out*() functions for secure data transfer between main memory and SPE's LS. *decrypt_in*() decrypts encrypted data in the main memory with a user defined shared application key, and copies it to the LS. *encrypt_out*() encrypts plaintext data in SPE's LS with a user defined shared application key, and copies the encrypted data to the main memory.

4.6 Performance Evaluation

This section explores the performance impact of the proposed hardware-based security features. First, the performance of the parallelized non-secure algorithm on the PlayStation3 is evaluated. Then, the performance of both the secure K-Means clustering algorithm and the non-secure K-Means clustering algorithm on the Cell secure system simulator are compared. Finally, the performance overhead for security is discussed.

4.6.1 Evaluation Environment

In this section, all the evaluations of the secure K-Means clustering are done on top of the Cell secure system simulator version 2.1. It is because of the fact that the hardware security features are not accessible on the physical hardware with the current Cell security SDK. The algorithms are evaluated in the cycle-accurate mode to gather and compare performance statistics. For simplicity, a 3.2GHz Cell with eight SPEs and 25.6GB/s of memory bandwidth is chosen. The Cell processors in IBM QS20 Cell Blade Server and the PlayStation3 have the same configuration. The evaluations of the non-secure K-Means

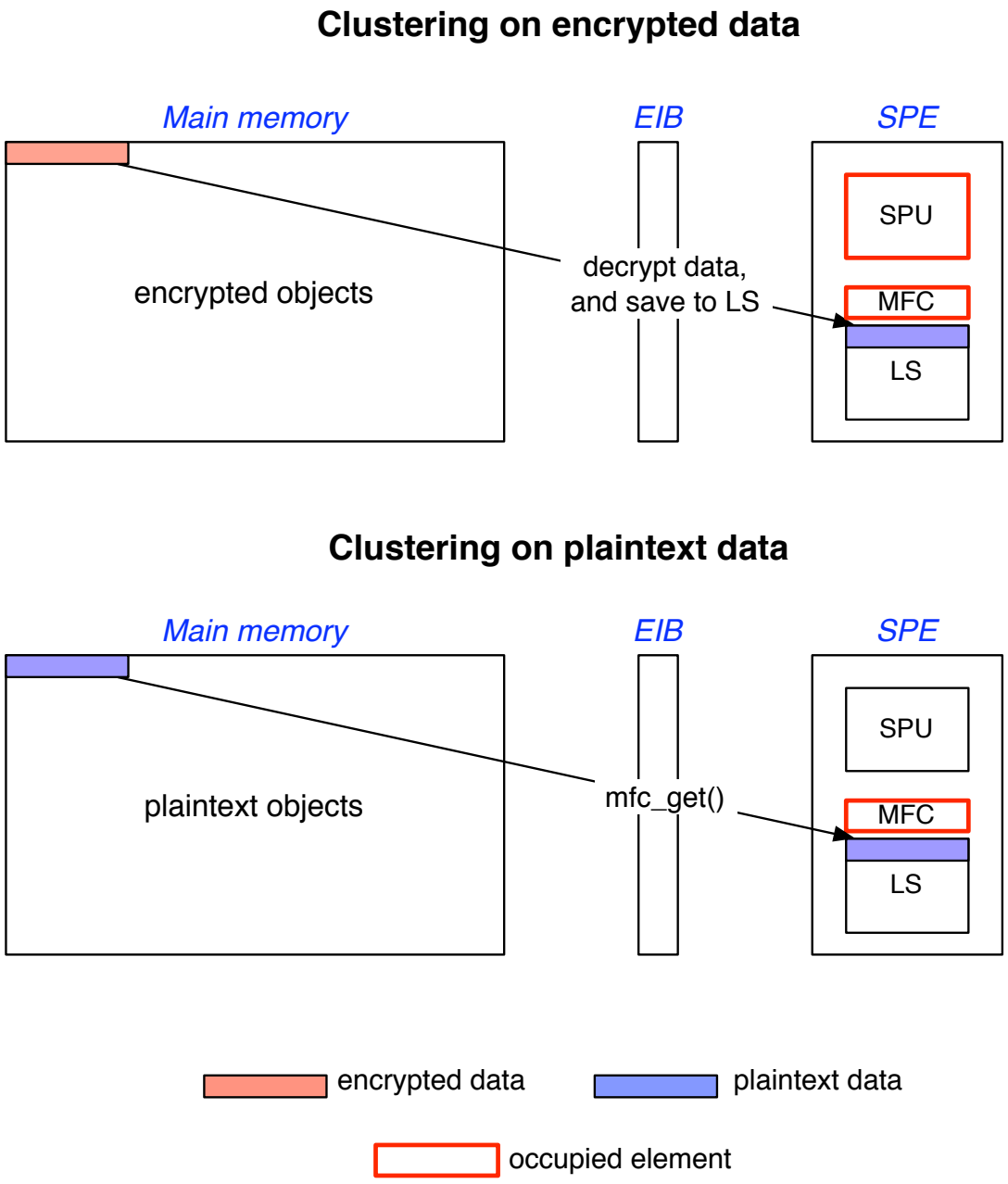


Figure 4.10: *Difference between plaintext data and encrypted data for clustering.*

clustering are done on both PlayStation3 (six SPEs) and the same simulator.

To study the overhead for security features, SPE performance statistics of both the secure K-Means clustering and the non-secure K-Means clustering is gathered at the runtime. The detail process statistics of the different process stage are also gathered, including data transfer cycles and buffer processing cycles. The performance is evaluated with the IBM Cell BE Security SDK CDA Version 2.1. On the PlayStation 3, the non-secure K-Means clustering is evaluated with the IBM Cell BE SDK Version 2.1. Table 4.1 lists the specification of the evaluation environment.

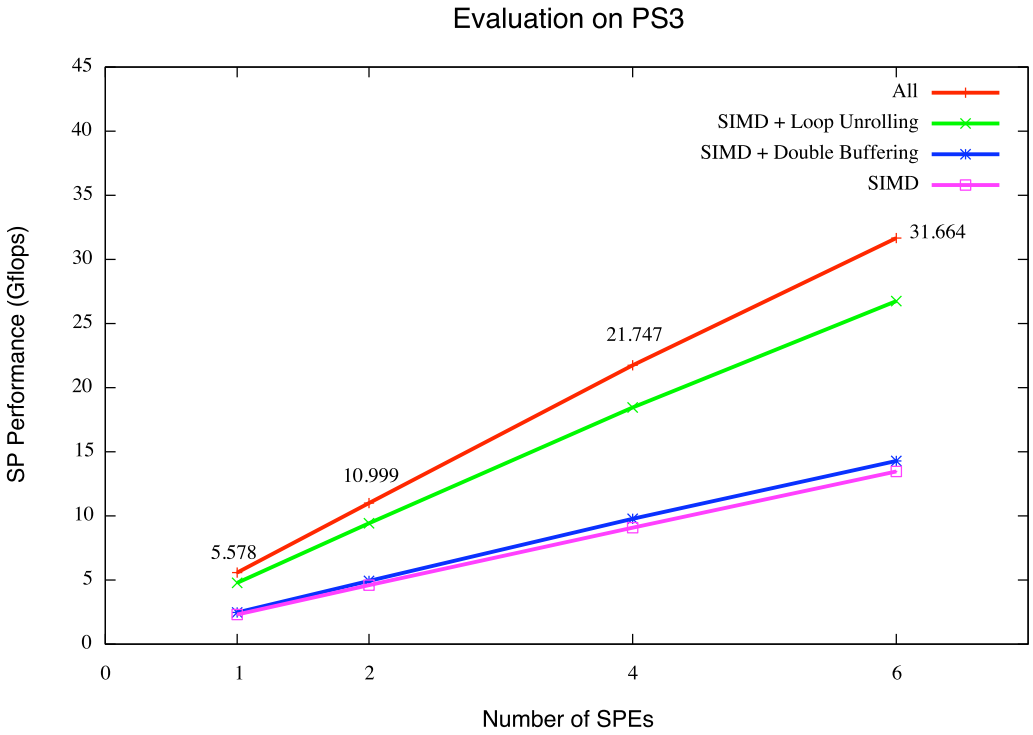
Table 4.1: *Evaluation environment for the secure and non-secure K-Means clustering.*

Software	
OS	Fedora Core 5 PPC
SDK	IBM Cell BE Security SDK Version 2.1 IBM Cell BE SDK Version 2.1 on PlayStation3

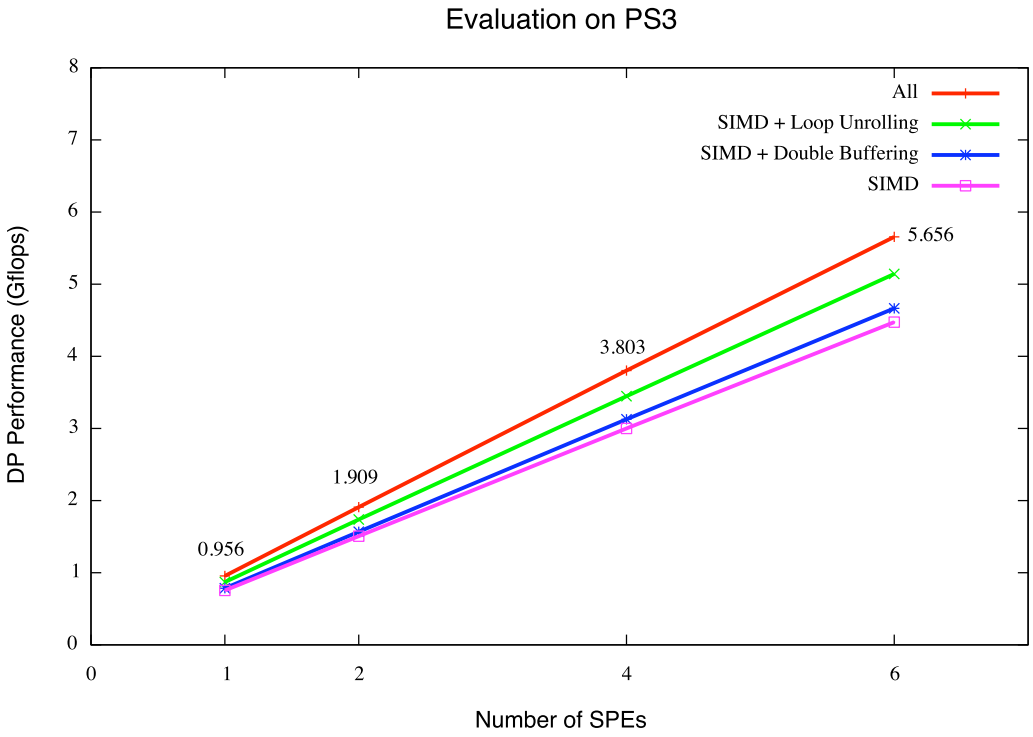
4.6.2 Effects of the Optimizations

Figure 4.11 compares the performance on PlayStation3, when using different optimization combinations. The data set to be clustered for evaluation has 16384 objects; the number of coordinates is 64. Each buffer stores up to 1024 elements, either single precision floating point or double precision floating point.

In Figure 4.11(a), *SIMD+Loop Unrolling* shows a significant improvement (about 2 times faster) over SIMD approach. *Double Buffering* provides about a 7% performance improvement without *Loop Unrolling*, and about a 17% improvement with *Loop Unrolling*. The improvement of *Double Buffering* without *Loop Unrolling* is not as remarkable as with *Loop Unrolling*. The reason is the high computation/communication ratio. Each data buffer's communication takes much fewer cycles than the computation. While *Loop Unrolling* is applied, the computation time is reduced to about half of the original SIMD code. Therefore, the *Double Buffering* is more efficient.



(a) Single Precision



(b) Double Precision

Figure 4.11: *Performance evaluation on PlayStation3.*

The evaluation results of double precision are shown in Figure 4.11(b). The optimizations are not as efficient as for single precision. The reason is that the pipeline hazard leads to 61.6% of the stalled cycles. The double precision instructions take 13 cycles on SPU. These instructions are partially pipelined (the last 7 cycles). Furthermore, no other instructions are dual-issued with double-precision instructions. *Loop Unrolling* gives 21.6% improvement with *Double Buffering*, and 15% improvement without *Double Buffering*. *Double Buffering* provides about 10% with *Loop Unrolling*, and about 4% without *Loop Unrolling*.

4.6.3 Performance Statistics of SPE Threads

Table 4.2 shows a dynamic timing analysis of both the secure K-Means clustering and the non-secure K-Means clustering on the same data set. In the previous section, the performance is presented in terms of "Gflop/s." Because of the massive computation cost for data decryption and lack of support for security features on the hardwares, it is not appropriate to discuss the performance of the secure K-Means clustering using "Gflop/s." The performance is discussed in terms of the performance statistics.

Table 4.2: *Performance statistics of SPE threads.*

Single Precision	Secure K-Means	Non-Secure
Total processing cycles (20 iterations)	342,559,460	35,644,760
Cycles for buffer transfer	287,501	9,364
Cycles for buffer clustering	51,759	51,600
Double Precision	Secure K-Means	Non-Secure
Total processing cycles (20 iterations)	744,896,330	242,251,574
Cycles for buffer transfer	287,965	9,373
Cycles for buffer clustering	172,806	172,693

For both single precision and double precision, the number of buffer transfer cycles for the secure K-Means clustering is 30.7x of the corresponding value for the non-secure K-Means clustering. The reason is the massive computation cost for data buffer decryption.

While plaintext data transfer from main memory to SPE's LS requires MFC to execute DMA commands, encrypted data decryption occupies both MFC for DMA transfer and SPU for data decryption. Moreover, the numbers of buffer transfer cycles for the secure K-Means clustering are 5.55x (single precision) and 1.67x (double precision) more than those of buffer clustering cycles. This means that the decryption process takes the most of cycles during the process. While the numbers of buffer clustering cycles for both two algorithms are almost the same, the huge overhead for decryption makes the non-secure K-Means clustering outperform the secure algorithm by 5.56x (single precision) and 2.53x (double precision), in terms of total buffer processing cycles.

Furthermore, since encrypted data decryption occupies both MFC for DMA transfer and SPU for data decryption, the double buffering optimization is not applicable to the secure K-Means clustering algorithm. The evaluation of the non-secure algorithm shows that double buffering provides about 17% (single precision) and 10% (double precision) improvements. The secure K-Means clustering algorithm cannot benefit from this optimization.

Because of the large computation cost for decryption and lack of double buffering, the overall processing speeds of the secure K-Means clustering for single precision and double precision are 10.4% and 32.5% of the original non-secure algorithm.

The number of double precision buffer clustering cycles is 3.33x more than that for single precision. Same as the non-secure algorithm, it is because of the lack of optimization for double precision instructions. Thanks to a relatively large number of needed buffer clustering cycles for double precision data, the performance overhead for double precision is less significant than the one for single precision.

Despite the huge performance degradation, the secure K-Means clustering algorithm running on the Cell processor still greatly outperforms the non-secure K-Means clustering algorithm running on the two commodity processors (Athlon64 3400+ and PowerPC G4 1.67GHz). As shown in Table 4.3, for single precision, the secure algorithm running on the Cell processor is 3.07x and 8.29x faster than the non-secure algorithm running on

the other two commodity processors, respectively. In the case of double precision, the secure K-Means clustering algorithm running on the Cell processor still outperforms the non-secure one on the commodity processors by 1.83x and 5.57x, respectively.

Table 4.3: *Performance of the K-Means clustering algorithm on the processors.*

	Secure algorithm	Non-secure algorithm	
	Cell	Athlon64	PowerPC
Single Precision (Gflops)	4.39	1.43	0.53
Double Precision (Gflops)	2.45	1.34	0.44

4.7 Conclusions

In this chapter, the related work in privacy preserving data mining has been discussed. Then, a hardware base secure data processing method for the volunteer computing platform has been proposed. After that, a widely available secure processor on the volunteer computing platforms - the Cell processor has been studied. A sample application - secure K-Means clustering has been designed on the Cell processor, on top of the hardware-based secure data processing method. The performance of the secure K-Means clustering has been evaluated on the Cell secure system simulator. The performance statistics has been compared with the corresponding statistics for the non-secure K-Means clustering on the same simulator. The results indicate that the secure data processing application on secure hardware greatly outperforms the non-secure K-Means clustering algorithm running on the general purpose CPU. However, a huge performance overhead was introduced by the decryption process of the security features.

Because of the PlayStation3, millions of the secure processors will be connected to the Internet. The high performance, the large number of peers in the future, and the security features make volunteer computing a promising solution for large volume sensitive data processing.

Chapter 5

Conclusions and Future Work

Volunteer computing is a promising technology that provides more computing power than any supercomputer, cluster or grid. Despite the massive computing power offered by the existing volunteer computing platforms, the applicable areas of existing volunteer computing platforms are limited to embarrassingly parallel computation on publicly available data. It is because of the two major issues with the existing volunteer computing platforms: lack of inter-task dependency support, and lack of security features. In this dissertation, a secure and high performance volunteer computing platform has been proposed. It targets on providing a widely applicable computing platform, and solves the two major issues of the existing volunteer computing platforms.

These issues exist on different layers of the volunteer computing platforms, including task management layer, resource management layer, and the application layer. Several solutions have been proposed in this dissertation to address these issues, including:

- Workflow management mechanism: extends the applicable areas. It is introduced on the task management layer to support complex inter-task dependency.
- Optimized task dispatch: mitigates the performance degradation introduced by the inter-task dependency. It is designed on the resource management layer to find optimal task dispatches. The worker availability checking is also added to gather

and provide required runtime information.

- Hardware-based secure data processing method: enables sensitive data processing on the volunteer computing. This secure data processing method involves both the dispatch and the worker on the application layer.

Chapter 2 has reviewed the P2P-RPC, workflow management mechanisms for grid systems, and a runtime task replication management for grid. Since none of the related work is suitable for large scale, highly volatile volunteer computing platforms, a dependable workflow management mechanism is desired. A workflow management mechanism has been proposed to handle inter-task dependency. A redundant task dispatch policy and its runtime optimization have been designed to guarantee high dependability of the workflow management mechanism on volunteer computing platforms. This dependable workflow management mechanism has been evaluated. The results prove that this redundant task dispatch policy guarantees high dependability, compared with the non-redundant dispatch policy.

Chapter 3 has discussed the limitation of the work in Chapter 2. Since the average failure probability model is not the best fitted for a real world volunteer computing peer, a heuristics-based mechanism has been proposed to estimate failure probability by gathering runtime resource availability data. Using this mechanism, the *Least Failure Probability Dispatch* policy has been designed to minimize the overall failure probability at the runtime, by considering the task assignment between multiple tasks and workers. To study the effectiveness of this policy, a simple extension of the original redundant task dispatch policy and a greedy dispatch policy have been introduced. The total process times of a computing job using different policies have been compared, using two real world availability trace data sets. The results prove the *LFPD* policy's advantage over the simple extension of the original redundant task dispatch policy. The results also show that the *LFPD* can beat the greedy dispatch with a small mean task process time. The effects of different parameters and different trace data's availability characteristics have

been discussed. Then, the *LFPD* policy has been simulated with and without the ability to identify different types of workers, using a trace data set consisting of two types of workers. The results indicate that by identifying different worker types, an additional performance improvement can be achieved.

Chapter 4 has explored the potential of secure volunteer data processing using a proposed hardware-based cryptography approach. The processing of rapidly increasing data volume requires massive computing power. The existing volunteer computing platform cannot process sensitive data, because of the security issues. The related work of privacy preserving data mining has been reviewed. Then, a cryptography-based secure data processing method has been proposed for volunteer computing. The advantage of using a hardware-based approach to implement this method has been discussed. A sample application and a widely available processor with hardware security features - the Cell processor have been used to study the potential of this method. The results demonstrate the huge performance advantage of an application with the hardware-based secure data processing method over the non-secure application running of general purpose CPUs. However, a great performance overhead for the data decryption has also been shown.

These proposed solutions have solved the issues on different layers of the existing volunteer computing platforms. Enhanced with these proposed solutions, a secure and high performance volunteer computing platform has been achieved. It has the ability to solve a large number of complex computational problems, on both publicly available data and sensitive data.

The cryptography-based secure data processing method for volunteer computing introduces a performance overhead caused by the data decryption. While the hardware-based security features make the secure processor compelling, the performance overhead becomes a new issue. This issue will be studied in the future work.

References

[1] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing*, 8 Nov. 2004, pp. 4–10.

[2] "The great internet mersenne prime search." [Online]. Available: http://www.mersenne.org

[3] "Distributed.net." [Online]. Available: http://www.distributed.net

[4] "Seti@home." [Online]. Available: http://setiathome.ssl.berkeley.edu

[5] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[6] "Folding@home," http://folding.stanford.edu/.

[7] "Berkeley open infrastructure for network computing." [Online]. Available: http://boinc.berkeley.edu

[8] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Néri, and O. Lodygensky, "Computing on large-scale distributed systems: Xtrem web architecture, programming models, security, tests and convergence with grid," *Future Generation Computer Systems*, vol. 21, no. 3, pp. 417–437, 2005.

[9] A. A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia: architecture and performance of an enterprise desktop grid system." *Journal of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 597–610, 2003.

[10] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Alchemi: A .net-based enterprise grid computing system." in *International Conference on Internet Computing*, 2005, pp. 269–278.

[11] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov, "Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment," in *Proceedings of Third International Workshop on Grid Computing*, 2002, pp. 1–12.

[12] "Project JXTA." [Online]. Available: http://www.jxta.org

[13] L. Gong, "JXTA: A network programming environment," *IEEE Internet Computing*, vol. 5, no. 3, pp. 88–95, May-June 2001.

[14] *JXTA v2.0 Protocols Specification.* [Online]. Available: http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.pdf

[15] "Folding@home client statistics by os," 2008. [Online]. Available: http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats

[16] "The 32th top500 list." [Online]. Available: http://www.top500.org/lists/2008/11

[17] I. Foster, *Designing and building parallel programs: concepts and tools for parallel software engineering.* Addison-Wesley, 1995. [Online]. Available: http://www-unix.mcs.anl.gov/dbpp/

[18] S. Djilali, "P2P-RPC: Programming scientific applications on peer-to-peer systems with remote procedure call," in *Proceedings of Third IEEE/ACM International Symposium on Cluster Computing and the Grid*, 12-15 May 2003, pp. 406–413.

[19] M. Sato, T. Boku, and D. Takahashi, "OmniRPC: a grid RPC system for parallel programming in cluster and grid environment," in *Proceedings of Third IEEE/ACM International Symposium on Cluster Computing and the Grid*, 12-15 May 2003, pp. 206–213.

[20] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi, "Ninf: A network based information library for global world-wide computing infrastructure," in *Proceedings of High-Performance Computing and Networking, International Conference and Exhibition, HPCN Europe*, 1997, pp. 491–502.

[21] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 171–200, September 2005.

[22] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, and G. Premchandran, "Webflow - a visual programming paradigm for web/java based coarse grain distributed computing," *Concurrency - Practice and Experience*, vol. 9, no. 6, pp. 555–577, 1997.

[23] H. P. Bivens, "Grid workflow," 2001, Grid Computing Environments Working Group, Global Grid Forum.

[24] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, "Gridflow: Workflow management for grid computing," in *Proceedings of Third IEEE/ACM International Symposium on Cluster Computing and the Grid*, 12-15 May 2003, pp. 198–205.

[25] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi, "GridAnt: A client-controllable grid workflow system," in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 5-8 Jan. 2004, p. 10pp.

[26] "Apache Ant." [Online]. Available: http://ant.apache.org

[27] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan,

Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. Rey-Cenvaz, "Programming the grid: Distributed software components, P2P and grid web services for scientific applications," *Cluster Computing*, vol. 5, no. 3, pp. 325–336, 2002.

[28] M. Lorch and D. G. Kafura, "Symphony - a java-based composition and manipulation framework for computational grids," in *Proceedings of Second IEEE/ACM International Symposium on Cluster Computing and the Grid*, 21-24 May 2002, pp. 136–143.

[29] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington, "Optimisation of component-based applications within a grid environment," in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, 10-16 Nov. 2001, pp. 30–47.

[30] F. Neubauer, A. Hoheisel, and J. Geiler, "Workflow-based grid applications," *Future Generation Computer Systems*, vol. 22, no. 1, pp. 6–15, 2006.

[31] S. Hwang and C. Kesselman, "A flexible framework for fault tolerance in the grid," *Journal of Grid Computing*, vol. 1, no. 3, pp. 251–272, 2003.

[32] J. H. Abawajy, "Fault-tolerant scheduling policy for grid computing systems." in *18th International Parallel and Distributed Processing Symposium*, 2004.

[33] A. Litke, D. Skoutas, K. Tserpes, and T. Varvarigou, "Efficient task replication and management for adaptive fault tolerance in mobile grid environments," *Future Generation Computer Systems*, vol. 23, no. 2, pp. 163–178, 2007.

[34] "W3C XML specification." [Online]. Available: http://www.w3.org/TR/REC-xml

[35] *OPNET Technologies Inc.* [Online]. Available: http://www.opnet.com/

[36] J. Gray, "A census of tandem system availability between 1985 and 1990," *IEEE Transactions on Reliability*, vol. 39, no. 4, pp. 409–418, Oct 1990.

[37] M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer, "Failure data analysis of a LAN of Windows NT based computers," in *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS 99)*, 1999, pp. 178–187.

[38] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems (USITS 03)*, 2003, pp. 1–1.

[39] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN 06)*, 2006, pp. 249–258.

[40] J. Aldrich, "R. A. Fisher and the making of maximum likelihood 1912-1922," *Statistical Science*, vol. 12, no. 3, pp. 162–176, 1997.

[41] J. K. Patel, C. H. Kapadia, and D. B. Owen, *Handbook of Statistical Distributions*. Marcel Dekker, Inc., 1976.

[42] A. Goel, "Software reliability models: Assumptions, limitations, and applicability," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 12, pp. 1411–1423, Dec. 1985.

[43] R. K. Iyer and D. J. Rossetti, "Effect of system workload on operating system reliability: A study on IBM 3081," *IEEE Transactions on Software Engineering*, vol. 11, no. 12, pp. 1438–1448, 1985.

[44] I. Lee, D. Tang, R. Iyer, and M.-C. Hsueh, "Measurement-based evaluation of operating system fault tolerance," *IEEE Transactions on Reliability*, vol. 42, no. 2, pp. 238–249, Jun 1993.

[45] M. W. Mutka and M. Livny, "Profiling workstations' available capacity for remote execution," in *Proceedings of the 12th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*, 1988, pp. 529–544.

[46] J. Plank and W. Elwasif, "Experimental assessment of workstation failures and their impact on checkpointing systems," *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing*, pp. 48–57, Jun 1998.

[47] J. Tian and Y. Dai, "Understanding the dynamic of peer-to-peer systems," in *Sixth International Workshop on Peer-to-Peer Systems (IPTPS 2007)*, Feb. 2007.

[48] M. Harchol-Balter and A. B. Downey, "Exploiting process lifetime distributions for dynamic load balancing," *ACM Transactions on Computer Systems*, vol. 15, no. 3, pp. 253–285, 1997.

[49] V. Paxson and S. Floyd, "Why we don't know how to simulate the Internet," in *Proceedings of the 29th conference on Winter simulation*, 1997, pp. 1037–1044.

[50] J. Xu, Z. Kalbarczyk, and R. Iyer, "Networked Windows NT system field failure data analysis," *Proceedings of 1999 Pacific Rim International Symposium on Dependable Computing*, pp. 178–185, 1999.

[51] D. Nurmi, J. Brevik, and R. Wolski, "Modeling machine availability in enterprise and wide-area distributed computing environments," in *Proceedings of the 11th International Euro-Par Conference*, 2005, pp. 432–441.

[52] A. Iosup, M. Jan, O. Sonmez, and D. Epema, "On the dynamic resource availability in grids," in *Proceedings of 8th IEEE/ACM International Conference on Grid Computing*, Sept. 2007, pp. 26–33.

[53] F. Nadeem, R. Prodan, and T. Fahringer, "Characterizing, modeling and predicting dynamic resource availability in a large scale multi-purpose grid," in *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID 08)*, 2008, pp. 348–357.

[54] J. Brevik, D. Nurmi, and R. Wolski, "Automatic methods for predicting machine availability in desktop grid and peer-to-peer systems," in *Proceedings of the 2004*

IEEE International Symposium on Cluster Computing and the Grid (CCGRID 04), 2004, pp. 190–199.

[55] M. Litzkow, M. Livny, and M. Mutka, "Condor - a hunter of idle workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems*, 1988, pp. 104–111.

[56] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[57] D. Long, A. Muir, and R. Golding, "A longitudinal survey of internet host reliability," in *Proceedings of the 14TH Symposium on Reliable Distributed Systems (SRDS 95)*, 1995, pp. 2–9.

[58] "Grid'5000 project." [Online]. Available: https://www.grid5000.fr

[59] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche, "Grid'5000: a large scale and highly reconfigurable experimental grid testbed." *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, Nov. 2006.

[60] "Austrian grid." [Online]. Available: http://www.austriangrid.at/

[61] X. Ren and R. Eigenmann, "Empirical studies on the behavior of resource availability in fine-grained cycle sharing systems," in *Proceedings of 2006 International Conference on Parallel Processing*, 2006, pp. 3–11.

[62] B. Rood and M. Lewis, "Multi-state grid resource availability characterization," in *Proceedings of 8th IEEE/ACM International Conference on Grid Computing*, Sept. 2007, pp. 42–49.

[63] J. W. Mickens and B. D. Noble, "Exploiting availability prediction in distributed systems," in *Proceedings of the 3rd conference on 3rd Symposium on Networked Systems Design & Implementation (NSDI 06)*, 2006, pp. 6–19.

[64] "OMNeT++." [Online]. Available: http://www.omnetpp.org

[65] S. Guha, N. Daswani, and R. Jain, "An experimental study of the Skype peer-to-peer VoIP system," in *The 5th International Workshop on Peer-to-Peer Systems*, 2006. [Online]. Available: http://saikat.guha.cc/pub/iptps06-skype.pdf

[66] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs," *ACM SIGMETRICS Performance Evaluation Review*, vol. 28, no. 1, pp. 34–43, 2000.

[67] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, second edition ed. Morgan Kaufmann, 2005.

[68] V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis, "State-of-the-art in privacy preserving data mining," *ACM SIGMOD Record*, vol. 3, no. 1, pp. 50–57, March 2004.

[69] J. Vaidya and C. Clifton, "Privacy preserving association rule mining in vertically partitioned data," in *Proceedings of the Eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 639–644.

[70] M. Kantarcioglu and C. Clifton, "Privacy-preserving distributed mining of association rules on horizontally partitioned data," in *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2002.

[71] W. Du and Z. Zhan, "Building decision tree classifier on private data," in *IEEE ICDM Workshop on Privacy, Security and Data Mining*, vol. 14, 2002, pp. 1–8.

[72] Y. Lindell and B. Pinkas, "Privacy preserving data mining," *Journal of Cryptology*, vol. 15, no. 3, pp. 177–206, 2002.

[73] J. Vaidya and C. Clifton, "Privacy-preserving k-means clustering over vertically partitioned data," in *Proceeding of the ninth ACM SIGKDD international conference on Knowledge discovery in data mining*, 2003, pp. 206–215.

[74] V. S. Verykios, A. K. Elmagarmid, E. Bertino, Y. Saygin, and E. Dasseni, "Association rule hiding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 4, pp. 434–447, 2004.

[75] Y. Saygin, V. S. Verykios, and C. Clifton, "Using unknowns to prevent discovery of association rules," *SIGMOD Record*, vol. 30, no. 4, pp. 45–54, 2001.

[76] D. Agrawal and C. C. Aggarwal, "On the design and quantification of privacy preserving data mining algorithms," in *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2001, pp. 247–255.

[77] R. Agrawal and R. Srikant, "Privacy-preserving data mining," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 439–450.

[78] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, "Privacy preserving mining of association rules," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 217–228.

[79] S. J. Rizvi and J. R. Haritsa, "Maintaining data privacy in association rule mining," in *Proceedings of the 28th international conference on Very Large Data Bases*, 2002, pp. 682–693.

[80] D. Lie, J. Mitchell, C. A. Thekkath, and M. Horowitz, "Specifying and verifying hardware for tamper-resistant software," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy (SP '03)*. IEEE Computer Society, 2003, pp. 166–177.

[81] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki,

M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, "The design and implementation of a first-generation cell processor," in *The International Solid-State Circuits Conference*, 2005, pp. 184–185.

[82] "A streaming processing unit for a cell processor," in *Digest of Technical Papers, The International Solid-State Circuits Conference*, 2005, pp. 134–135.

[83] *Cell Broadband Engine Programming Tutorial Version 1.1*, IBM developerWorks, 2006.

[84] K. Shimizu, D. Brokenshire, and M. Peyravian, "Cell broadband engine support for privacy, security, and digital rights management applications," IBM White Paper, Oct. 2005.

[85] *Cell Broadband Engine Security Software Development Kit 3.0 Installation and User's Guide Version 3.01*, IBM, September 2007.

[86] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

Publications

Journal

1. Hong Wang, Hiroyuki Takizawa and Hiroaki Kobayashi: A Performance Study of Secure Data Mining on the Cell Processor (extended version). *International Journal of Grid and High Performance Computing*, scheduled for Vol. 1(2), pp. 30-44, 2009.

2. Hong Wang, Hiroyuki Takizawa and Hiroaki Kobayashi: A Dependable Peer-to-Peer Computing Platform. *Future Generation Computer Systems*, Vol. 23(8), pp. 939-955, 2007.

Conference Papers

1. Hong Wang, Hiroyuki Takizawa and Hiroaki Kobayashi: A Performance Study of Secure Data Mining on the Cell Processor. *In Proceedings of 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'08)*, pp. 633-638, 2008.

2. Ling Xu, Hong Wang, Hiroyuki Takizawa and Hiroaki Kobayashi: A Reliability Model for Result Checking in Volunteer Computing. *In Proceedings of 2008 International Symposium on Applications and the Internet (SAINT'08)*, pp. 201-204, 2008.

3. Hong Wang, Hiroyuki Takizawa and Hiroaki Kobayashi: An Estimation-Based Redundant Task Dispatch Policy for Volunteer Computing Platforms. *In Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pp. 348-349, 2007.

4. Hong Wang, Hiroyuki Takizawa and Hiroaki Kobayashi: Evaluation of K-Means Clustering on the Cell Processor. *In Proceedings of High Performance Computing Symposium 2007 (HPCS'07)*, pp. 161-168, 2007.

5. Hong Wang, Hiroyuki Takizawa and Hiroaki Kobayashi: A Workflow Management Mechanism for Peer-to-Peer Computing Platforms. *In Proceedings of Third International Symposium of Parallel and Distributed Processing and Applications (ISPA'05)*, pp. 827-832, 2005.